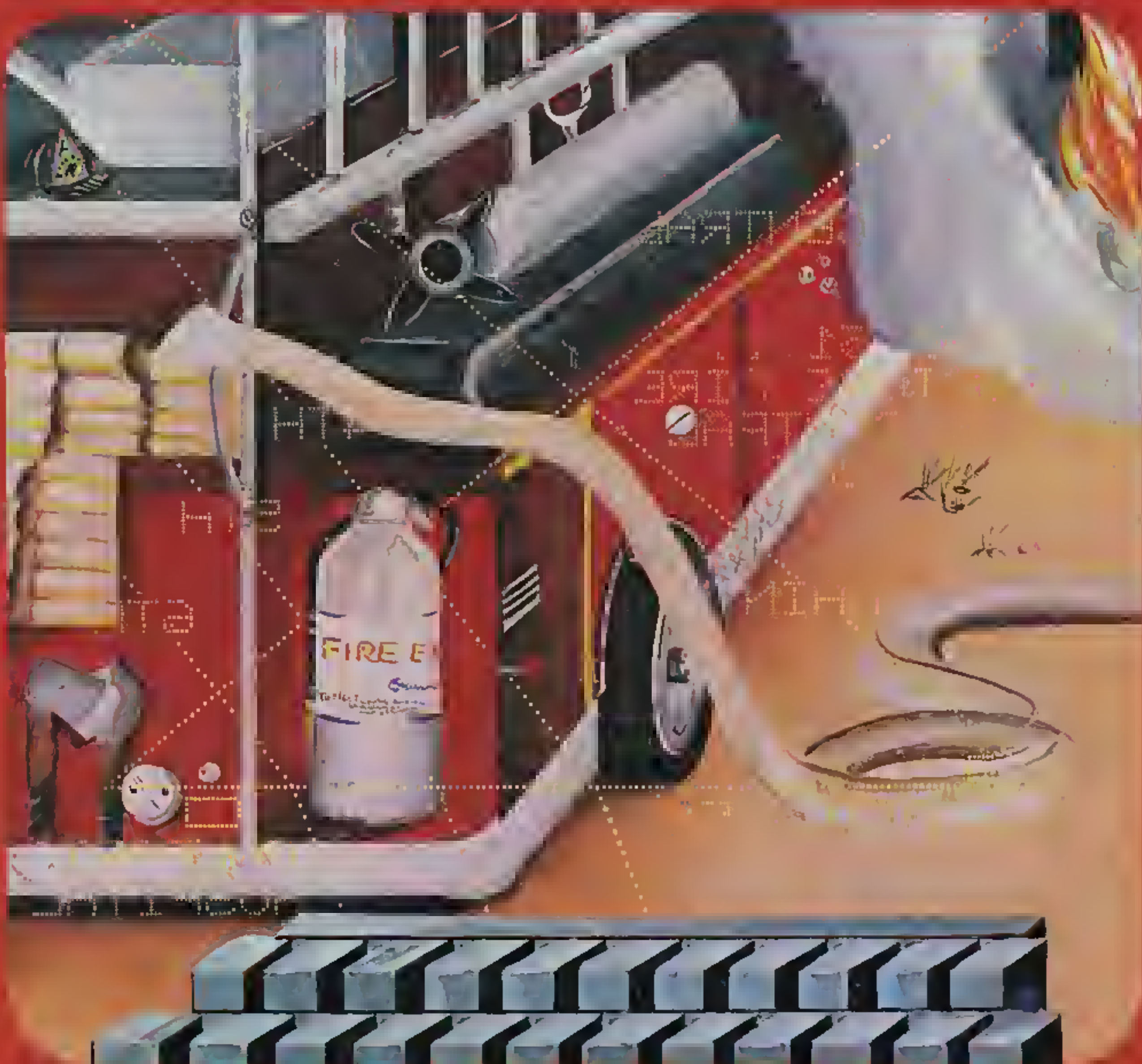
# MICRO

THE 6502 JOURNAL




JULY 1980

# MICRO™

## July 1980
## Issue Number 26




### Staff

**Editor/Publisher**
Robert M. Tripp

**Associate Editor**
Mary Ann Curtis

**Assistant Editor/ Advertising Manager**
L. Catherine Bland

**Circulation Manager**
Carol A. Stark

**Technical Assistant**
Theresa MacMaster

**Art/Advertising Coordinator**
Terry Spillane

**Comptroller**
Donna M. Tripp

MICRO™ is published monthly by:
MICRO INK, Inc., Chelmsford, MA 01824
Second Class postage paid at:
Chelmsford, MA 01824
Publication Number: COTR 395770

| | |
|---|---|
| Subscription rates: U.S. | $15.00 per year |
| Foreign surface mail | $18.00 per year |
| Central America air | $27.00 per year |
| Europe/Japan air | $33.00 per year |
| Other air mail | $39.00 per year |

For back issues, subscriptions, change of address or other information, write to:
MICRO, P.O. Box 6502,
Chelmsford, MA 01824
or call
617/256-5515

# A MICRO Potpourri

While cleaning out my desk, as part of adding office space to MICRO, I uncovered a vast cache of notes that I had written to myself: little things which I wanted to pass on to MICRO's readers.

**Canadian Mail:** There seem to be problems with the Canadian mail service. In recent months we have been receiving more reports of non-delivery from our northern neighbors than from all of the US subscribers. We hope that the service gets better, and for now can only counsel patience. If you magazine does not reach you by the middle of the month, then complain to your postal service.

**Mailing Date:** **MICRO** is always in the mail before the first of the issue month. The actual mailing date varies as a function of the month, but is generally between the 24th and 28th. The Second Class mail, in the US, is *supposed* to get to all points within a week.

**Limerick Contest:** Since I have been declared ineligible by my staff to officially enter the **MICRO** limerick contest [a most unfair rule I think], I am going to excercise editorial perogative, if not editorial judgment, and present it here!

*A clever programmer named Mike Rowe,*
*Said, "I get double use from each MICRO.*
*First I learn what to do*
*With my Sixty-Five-Oh-Two,*
*Then I use it to paper train my crow!*

[Now, don't you just ***know*** that you can do better than that? Only a few weeks left to get your entry in.]

**Mike Rowe:** The first issue of **MICRO**, in October 1977, contained the following 'biographical' notes about Mike Rowe: 'He prefers hexadecimal notation since he has eight fingers on each hand', and is a 'Computer consultant for the Starship Enterprise'. Apparently some readers missed the first issue, and/or have never said the name out loud and discovered the hidden meaning. Mike Rowe is, of course, the name used to indicate that an article has been prepared by one or more members of the MICRO Staff from material supplied by others. The Software Catalog is an example. We have been surprised at the amount of mail we get addressed to Mike Rowe. Since 1977 we have discovered at least three others: Michael Roe — a subscriber; Mike Rowe Productions — also a subscriber; and Mike Rowe who, according to the newspaper, is the best stock car driver in Maine. If you happen to know of any other 'Mike Rowe', we would like to hear about him.

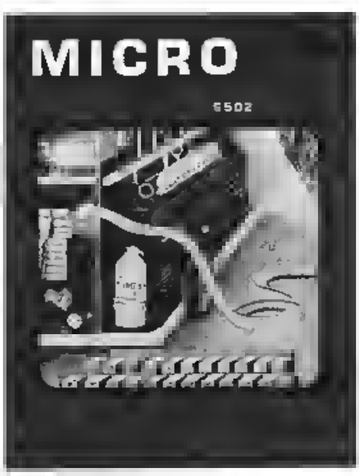
**MICRO Advertising and Advertisers:** Advertising is very important to MICRO for two reasons: first, it provides some very important and timely information about what is available, and, second, it supports the magazine. The reason that MICRO has been able to grow from 28 to 84 pages, has been due to the terrific support of the advertisers. We hope this will continue to grow. You can help. All it takes is informing an advertiser that you 'Saw It in MICRO'. That's all. Advertisers do not generally have any simple way to determine the effectiveness of a particular ad. Feedback from the buying public is the most effective way of telling an advertiser that his ad is working. So, when you place an order, please mention MICRO.

**Reader Feedback:** The advertisers are not the only people interested in hearing from you. The MICRO staff welcomes reader feedback. What types of material do you like best? Which articles have been most useful? Do you like the format? How about the three-hole punch? What new features would you like to see? Let us know. We want to keep MICRO serving its readership effectively.

**Writing for MICRO:** MICRO pays top rates for articles. If you have good 6502 related ideas, programs, etc., consider writing about them for MICRO. We have prepared a **MICRO Writer's Guide** to help. For your copy, simply send a self-addressed [we'll provide the stamp] envelope requesting the guide.

**Free MICRO:** If you are a subscriber and know someone who should be receiving MICRO [like the guy who keeps 'borrowing' your copy], send us his name and address along with your subscription label. We will send one sample copy. Since this does involve a fair amount of time and [illegible], we would appreciate your only sending in names of people who either own 6502 equipment or who you feel are seriously interested in the 6502 world. This offer expires August 15, 1980, so do it now.

Robert M. Tripp




## Emergency MICRO

**Cover Artist**
**Terry Spillane**

***Graphic Data Retrieval Systems***

This month's cover shows one type of Graphic Data Retrieval System: a fire department system to keep track of the equipment available for meeting various emergency conditions. While the concept is not new or specific to micros, it is a technique which can have broad application and which is quite suited to the display oriented microcomputers.

A GDRS basically combines graphic data, such as a map, with alphanumeric data. In the cover example, a map of the section of the city which contains an emergency condition, in this case a fire, is displayed to quickly show the operator the locations of relevant resources: a fire station, hospital, police, ambulance, etc. The status of each potential resource is given as alphanumeric information. As the operation progresses, this information can be continually updated either manually via a keyboard or, in a fancy system, automatically via various devices which would track the vehicles.

This is a very dramatic example of the technique. Many other less dramatic but nonetheless important uses can be conceived for GDRS technique. The flow of material through any process, from an oil pipe line to a auto production facility, can be tracked and displayed. The operator can 'zoom in' on any particular part of the operation which is of interest. The program can automatically display whatever portions of the process are most critical at any time.

One of the nice aspects of performing a GDRS task on a micro is that the graphics do not generally have to be very fancy. A simple set of character graphics: horizontal, vertical, and diagonal lines, can usually provide all of the detail necessary.

The GDRS method can be used to solve many different types of problems. Think about it application in your areas of interest. It can be an effective and efficient method.

# SYM-1 Memory Search and Display

**Add these two naw commands to your SYM Monitor. They make it aasy to locate any string in mamory and provide a means to display data as ASCII whan dasired.**

**Nicholas Vrtis**

Here are two more extensions for the SYM monitor. They are relocatable, and do not use any memory other than that normally used by the monitor. I decided to write these two software "tools" because I kept needing them and no one else seemed to be writing anything close to what I needed. The memory search routine was written because I needed some easy way to find locations in programs after I have relocated them. I don't have a printer, so after I have made a couple of patches and moves, it is sometimes difficult to find a particular place in the program. The command has also turned out to be helpful when you have to find references to a particular address so you can change it, as I had to do when I got the new monitor ROM.

The memory display routine was developed because I needed some easy way to look at messages, source lines, and other character type data in memory. This was especially true when I started working on a Tiny Basic Intermediate Language Assembler some time ago. The SYM monitor just doesn't have any way of displaying memory as characters instead of hex digits, and I have trouble recognizing ASCII as two hex digits.

The memory search routine will handle up to an eight byte search argument. This is normally entered in hex after the prompt from the routine. If you want, you can enter a slash instead of the two hex digits. This indicates a "wild" character to the program. The definition of a "wild" character is that the position is counted, but any character is a valid match. This does not mean that you can't search for a slash character. The program will look for a slash if you enter it in hex as $2F. This means that the search argument "20/OC" will find the first occurance of any jump to a subroutine on page $OC, but "202FOC" would only find a jump to the subroutine at $C2F. This neat little programming trick is accomplished with a "byte used bit map" (how's that for a three dollar phrase?). In simple terms, each bit is SCPBUD corresponds to one byte in SCPBUF where the search key is saved. If the bit is on, it indicates that a "wild" character was entered in that position. A zero indicates a normal character. The distinction between a slash and $2F is actually made by INBYTE. The slash is non-hex, so INBYTE returns with the carry set. If the overflow is set, then the second character was the non-hex and it is an error. If the equal is set the character was the carriage return, and the program uses that to mean the end of the search argument. Finally, if none of the above is true, then the character that was entered is compared to a slash (INBYTE conveniently leaves the character in 'A'). For the slash, the carry is rolled into the bit map, setting the bit to a one. For normal hex bytes entered, the carry is clear on return from INBYTE, so when the rotate is done, a zero is set into the bit. The only other check made on input is to watch for more than eight bytes being entered. The beeper is beeped, and the character is ignored once eight have been accepted.

To perform the search, the program moves the bit map to a work area, since it will be destroyed in the process of the search. Each time we want to make a comparison between the key and memory, we first rotate one bit from the bit map work area into the carry. If the carry is set after the rotate, then the bit was on, and the program just pretends it got an equal compare. If the carry wasn't set, then the search byte is compared to memory for an equal. Simple, isn't it? Each time an unequal is found, the search address is incremented, and the search starts from position one of the key again.

Once the search argument is found, it is simple to output the address and then the data from memory (not from the search key, since it has the slashes in it).

There are a couple of not so obvious points to mention. The input search key, the key length, and the bit map are retained in the SYM RAM scope buffer area. This means some good news, and some bad news. The good news is that provided you don't do any output to the LED's, the argument will still be there the next time you use the routine. Since the U4 option with no parameters entered starts at the last used location plus one, using this option and entering a carriage return immediately for the search key will find the next occurance of that string. The bad news is that the routine won't work if you

are using the hex keypad for entry. Actually, the three parameter option will work since it doesn't do any I/O until after it has hit the end of memory, or found the string. The problem is that any time you do output to the LED's, that character also gets rotated into the scope buffer area, so the process of entering the search key shifts it over. If you are using the hex keypad and want to use the search routine, you will have to supply a 10 byte work area someplace else.

Finally, the value of "end of memory" is set to $0F at location $211 for my 4K system. If you have more or less memory, set this to the highest page number in your system.

As I mentioned earlier, the memory display routine is primarily designed for displaying ASCII type information. It has also turned out to be somewhat useful as a normal memory display since it displays more bytes per line than does the Verify command. Another advantage is that it ends with the "OLD" address pointing to the next location after the last one displayed. This means that repeated calls to the command without any argument will continue displaying memory.

The display format is a typical dump format. Sixteen bytes of data are displayed, first in hex, and then as alpha. Before the alpha is output, though, it is checked to make sure that it is a displayable character. As written, this program translates control characters, lower case character, and anything with the high order bit on, to an underscore. On some terminals this will display as the back-arrow. The purpose is to occupy a position with displayable characters so you can count how many characters in you are from the start of the line. If your terminal will display lower case, you may want to change location $30C to the highest displayable character for your terminal (lower case z is $7A). I would not recommend by-passing the translation of the control characters. At best, most terminals don't even print a space in their place, and at worst, they do unexpected things which make reading the line difficult.

For those of you who have put up the other monitor extensions from my article in the August issue, these routines can be added very easily. Simply change the address in the JMP U1 instruction that was at $237 in the listing, to a JMP U4 where U4 is the address that the new routines are moved to. Then change this program at $2AE to insert a JMP U1 in place of the SEC-RTS-NOP, and presto!—you have two new extensions. Both routines U4 and U5 are relocatable, so you don't have to bother running them through Relocate. Just block move them to where you want them. I moved them to the front of the Execute setups so I wouldn't have to learn a new starting address.

For those of you who didn't read the article, I will review some of the comments about how to extend the SYM monitor. First let me say that these routines are relocatable, with the only provision being that they must be in the same relative position to each other, or the branch at $268 will have to be adjusted. If you decide to only use U4, change the above location to a SEC-RTS ($3860). The U5 routine will operate by itself without any changes. As I mentioned before, these routines use only those memory locations normally reserved for the monitor, so they shouldn't conflict with your existing programs. Nor will they affect the operation of any of the SYM commands, with the exception that the 'OLD' address that is referred to in the manual will get changed by these commands in additon to the standard commands.

The SYM monitor vectors all "unrecognized" commands via a RAM vector located aat $A66D. The monitor considers anything it isn't programed for as unrecognized. Normally, $A66D points to an SEC-RTS sequence. This indicates to the routine ERMSG that the ER xx message should be printed. By the way, the xx is the hex digits for what is in 'A' when ERMSG is called, so this makes a handy error routine for your oun programs. Since SYNERTEK was nice enough to put this vector in Ram, it can be changed. Specifically in our case if it is changed to point out the starting address of U4, the monitor will branch there instead of to the SEC-RTS. If you will note, these routines execute and SEC-TRS whenever they encounter an error, or the command is not the cash value for U4 or U5. For a normal return, they have to make sure the carry is clear or the error message would get printed.

The monitor routines used in these programs are normal labels as defined in the cross reference listing for the monitor. In order to possibly save some of your sanity when you look at the code, I will mention that the parameter input areas are not numbered the way you would expect. The monitor always accepts input into the P3 area, and each time a new parameter is entered, it shifts the whole area up 16 bits. This means that if only one parameter is entered, it ends up in the P3 area, not in P1 as you would expect. For two parameters, the first parameter is in P2, and the second in P3. For three parameters, the numbers come out right. It gets sort of confusing the first time you try to figure it out, and those are not memory locations you can use any of the commands to look at, since the monitor zeros them out at the start of each command.

These routines were written for version 1.1 of the SYM monitor, which is a little different from version 1.0. In V1.0, both unrecognized commands and syntax errors (i.e. non-hex digits) were vectored through $A66D, not just the unrecognized commands as in V1.1. This means that if you have V1.0 you have to check to make sure that you are not there because of a syntax error. In order to make these work for version 1.0, insert the following just before U4 and make it the address that goes into $A66D:

```
CD 57 A6        CMP LSTCOM
See if command terminated properly
F0 02        BEQ  U4    Branch if OK
38        SEC  Else set the error flag
60        RTS  and return to the monitor
```

This will take care of things for both U4 and U5. People who already have my other extensions up won't have to bother, since UO already check for this condition before it does anything else.
The sixteen bit checksum for $200-$31F is $8F1B.

μ

```
LOC    ----OBJECT----    STMT

                         00001 ********************************************************************
                         00002 * SYM-1 USER MONITOR FUNCTION EXTENSIONS                            *
                         00003 *                                                                   *
                         00004 * U4 - MEMORY SEARCH                                                *
                         00005 *  0 PARMS        SEARCH FROM 'CURAD+1' TO 'END OF MEMORY'          *
                         00006 *  1 PARM         SEARCH FROM PARM 1 TO 'END OF MEMORY'             *
                         00007 *  2 PARMS        SEARCH FROM PARM 1 TO PARM 2                      *
                         00008 *  3 PARMS        SEARCH FOR PARM 1 AS ADDRESS FROM PARM 2 TO PARM 3 *
                         00009 * U5 - DISPLAY ALPHA MEMORY                                         *
                         00010 * 0 PARMS         DISPLAY 1 LINE FROM 'CURAD'                       *
                         00011 * 1 PARM          DISPLAY 1 LINE FROM PARM 1                        *
                         00012 * 2 PARMS         DISPLAY FROM PARM 1 TO PARM 2                     *
                         00013 *                                                                   *
                         00014 ********************************************************************

LOC    ----OBJECT----    STMT        SYM-1 MONITOR EXTENSION ROUTINES

0000                     00016          TEXT   ***** OBJECT CODE OF SYM-1 MONITOR EXTENSIONS *******
0000                     00017          TEXT   ***** INITIALIZATION MONITOR COMMANDS         *******
0200                     00018          ORG    $200

                         00020 ********************************************************************
                         00021 * PAGE ZERO ADDRESS LOCATIONS                                       *
                         00022 ********************************************************************
                         00023 *
00EE                     00024 CURAD    EQU    $EE                      SYM-1 'OLD' ADDRESS
00EC                     00025 ADJUSTMT EQU    $EC                      SYM-1 SCRATCH PAGE ZERO AREA
00FC                     00026 PZSCR    EQU    $FC                      SYM-1 SCRATCH PAGE ZERO AREA

LOC    ----OBJECT----    STMT          U4 - MEMORY SEARCH SYM-1 MONITOR EXTENSION

0200                     00028          TEXT  ********** U4 MEMORY SEARCH SYS-1 EXTENSION *********
                         00029 *********************************************************************
                         00030 * U4 MONITOR EXTENSION FOR THE SYM-1 -- MEMORY SEARCH                *
                         00031 *********************************************************************
                         00032 *                                                                    *
                         00033 * BY: N. VRTIS  - LSI/CCSD                                           *
                         00034 *                                                                    *
                         00035 * ACTION DEPENDS ON NUMBER OF PARMS ENTERED                          *
                         00036 *  0 PARMS       SEARCH FROM 'CURAD+1' TO 'END OF MEMORY'            *
                         00037 *  1 PARM        SEARCH FROM PARM 1 TO 'END OF MEMORY'               *
                         00038 *  2 PARMS       SEARCH FROM PARM 1 TO PARM 2                        *
                         00039 *  3 PARMS       SEARCH FOR PARM 1 AS ADDRESS FROM PARM 2 TO PARM 3  *
                         00040 *                                                                    *
                         00041 * NOTES:                                                             *
                         00042 * 1)       EXCEPT FOR THE 3 PARM OPTION, A SLASH ENTERED AS A SLASH  *
                         00043 *          INSTEAD OF HEX 2F WILL BE CONSIDERED A 'WILD CHARACTER'   *
                         00044 * 2)       IF NO SEARCH ARGUEMENT IS ENTERED, THE VALUES FROM LAST TIME *
                         00045 *          WILL BE USED.                                             *
                         00046 * 3)       THIS ROUTINE USES THE SYSTEM RAM SCOPE BUFFER FOR SCRATCH *
                         00047 *          AREA, SO IT WILL ONLY WORK WITH A TERMINAL UNLESS ANOTHER *
                         00048 *          SCRATCH AREA IS PROVIDED.                                 *
                         00049 *                                                                    *
                         00050 *********************************************************************

0200   C9 18             00052 U4       CMP    #$18                     CHECK FOR 'U4' COMMAND
0202   D0 64             00053          BNE    TOU5                     TRY NEXT COMMAND IF NOT THIS ONE
                         00054 *
0204   E0 01             00055          CPX    #1                       SEE HOW MANY PARMS
0206   90 08             00056          BCC    NOPRMS                   BRANCH IF ZERO PARMS
0208   D0 17             00057          BNE    TRYNXT                   OR IF MORE THAN 1
020A   20 A7 82          00058          JSR    P3SCR                    MOVE STARTING ADDRESS TO 'CURAD'
020D   20 BE 82          00059          JSR    DECCMP                   BACKUP 1 SO INC COMES OUT OK
0210   A9 0F             00060 NOPRMS   LDA    #$0E                     ZERO PARMS - USE HIGHEST RAM ADDR.
0212   8D 4B A6          00061          STA    P3H
0215   CE 4A A6          00062          DEC    P3L                      MAKE LOW ORDER = $EE
0218   20 B2 82          00063          JSR    INCCMP                   BUMP CURAD BY ONE
021B   F0 20             00064          BEQ    GETARG                   GET SEARCH IF NOT AT END
021D   90 1E             00065          BCC    GETARG
021F   18                00066          CLC                             DONE IF ALREADY TO END OF MEMORY
0220   60                00067          RTS                             RETURN TO MONITOR
0221   8A                00068 TRYNXT   TXA                             PRESERVE # OF PARMS ENTERED
0222   20 9C 82          00069          JSR    P2SCR                    P2 AREA HAS 'FROM' FOR 2 & 3 PARMS
0225   C9 02             00070          CMP    #2                       CHECK WHICH
0227   F0 14             00071          BEQ    GETARG                   GET ARGUEMENT IF 2 PARMS
                         00072 *
0229   AD 4E A6          00073          LDA    P1L                      ELSE MOVE PARM1 TO SEARCH KEY
022C   8D 00 A6          00074          STA    SCPBUF
022F   AD 4F A6          00075          LDA    P1H
0232   8D 01 A6          00076          STA    SCPBUF+1
0235   A9 00             00077          LDA    #$00                     INDICATE BYTES USED
0237   85 FC             00078          STA    PZSCR
0239   A2 02             00079          LDX    #2                       MAKE X=2 AS LENGTH
LOC    ----OBJECT----    STMT          U4 - MEMORY SEARCH SYM-1 MONITOR EXTENSION
023B   D0 20             00080          BNE    GOTCR                    AND WE HAVE THE PARM IN NOW
                         00081 *
023D   A2 00             00082 GETARG   LDX    #0                       NUMBER OF BYTES ENTERED
023F   86 FC             00083          STX    PZSCR                    SET TO ALL BYTES USED
0241   20 4D 83          00084          JSR    CRLF                     START ON A NEW LINE
0244   A9 3E             00085          LDA    #'>'                     DISPLAY PROMPT
0246   20 47 8A          00086          JSR    OUTCHR
0249   20 D9 81          00087 GALOOP   JSR    INBYTE                   GET SEARCH INPUT
024C   B0 0C             00088          BCS    NONHEX                   NEED TO CECH FOR / IF NON-HEX
024E   E0 08             00089          CPX    #8                       SEE IF GOT ENOUGH ALREADY
0250   B0 10             00090          BCS    BADY                     CAN'T HANDLE ANY MORE
0252   9D 00 A6          00091          STA    SCPBUF,X                 ELSE SAVE THE BYTE
0255   26 FC             00092 ROLLIN   ROL    PZSCR                    ROLL CARRY INTO BYTES USED
0257   E8                00093          INX                             BUMP FOR NEXT ONE
0258   D0 EF             00094          BNE    GALOOP                   UNCONDITIONAL
025A   70 06             00095 NONHEX   BVS    BADY                     IF SECOND IS NON-HEX IT IS BAD
025C   F0 0C             00096          BEQ    GOTCR                    BRANCH IF IT WAS CARRIAGE RETURN
025E   C9 2F             00097          CMP    #'/'                     ELSE CHECK FOR A SLASH
0260   F0 F3             00098          BEQ    ROLLIN                   IF YES - CARRY IS SET FOR ROLLIN
```

```
0262  20 72 89   BADY    JSR  BEEP          ERROR CHARACTER BEEP THE BEEPER
0265  18                 CLC                CLEAR CARRY
0266  90 E1              BCC  GALOOP        TO FORCE BRANCH
                 *
0268  D0 40      TOU5    BNE  U5            JUST PASSING THROUGH ON WAY TO U5
                 *
026A  CA         GOTCR   DEX                SEE IF GOT ANY SEARCH CHARACTERS
026B  30 08              BMI  EACHST        BRANCH IF NOT
026D  8E 1F A6           STX  SCPSTL        ELSE SAVE STRING LENGTH
0270  A5 FC              LDA  PZSCR         MOVE BYTES USED TO HOLD AREA
0272  8D 1E A6           STA  SCPBUD
                 *
0275  AC 1F A6   EACHST  LDY  SCPSTL        START OF TAIL END OF STRING
0278  AD 1E A6           LDA  SCPBUD        MOVE BYTES USED MAP TO WORK AREA
027B  85 FC              STA  PZSCR
027D  66 FC      EACHCH  ROR  PZSCR         ROLL 1 BIT OF MAP INTO CARRY
027F  B0 07              BCS  ISMTCH        IF ON IT WAS A SLASH AND IS MATCHED
0281  B1 FE              LDA  (CURAD),Y     OTHERWISE
0283  D9 00 A6           CMP  SCPBUF,Y        COMPARE SEARCH KEY TO THIS BYTE
0286  D0 19              BNE  NOMTCH        BRANCH IF NOT A MATCH
0288  88         ISMTCH  DEY                GOT A MATCH - NEXT SEARCH CHAR
0289  10 F2              BPL  EACHCH        CONTINUE IF MORE IN STRING
                 *
028B  20 16 83           JSR  CRLFSZ        ELSE OUTPUT ADDRESS OF START
028E  20 42 83           JSR  SPACE
0291  C8                 INY                PUT Y BACK TO ZERO
0292  B1 FE      OUTLOP  LDA  (CURAD),Y     LIST THE CHARACTERS FOUND
0294  20 FA 82           JSR  OUTBYT
0297  C8                 INY
0298  CC 1F A6           CPY  SCPSTL
029B  90 F5              BCC  OUTLOP
029D  F0 F3              BEQ  OUTLOP        DON'T FORGET THE LAST BYTE
029F  18                 CLC                CLEAR CARRY
02A0  60                 RTS                AND RETURN TO MONITOR
                 *
02A1  20 B2 82   NOMTCH  JSR  INCCMP        NO MATCH -BUMP TO NEXT START ADDRESS
02A4  90 CF              BCC  EACHST        CONTINUE SEARCH IF MEMORY LEFT
02A6  F0 CD              BEQ  EACHST
02A8  18                 CLC                ELSE RETURN TO MONITOR
02A9  60                 RTS
02AA                     TEXT ********** U5 - DISPLAY MEMORY SYM-1 EXTENSION ******
                 ***********************************************************************
                 * U5 MONITOR EXTENSION FOR THE SYM-1 -- DISPLAY ALPHA MEMORY          *
                 ***********************************************************************
                 *                                                                     *
                 * BY: N. VRTIS - LSI/CCSO                                             *
                 *                                                                     *
                 * ACTION:       SIMILAR TO SYM 'VERIFY', EXCEPT 'DLO' POINTS TO NEXT  *
                 *               ADDRESS AFTER THE COMMAND.                            *
                 * 0 PARMS       DISPLAY 1 LINE FROM 'CURAD'                           *
                 * 1 PARM        DISPLAY 1 LINE FROM PARM 1                            *
                 * 2 PARMS       DISPLAY FROM PARM 1 TO PARM 2                         *
                 *                                                                     *
                 ***********************************************************************

02AA  C9 19      U5      CMP  #$19          CHECK FOR U5 HASH CODE
02AC  F0 03              BEQ  U5STRT        BRANCH IF YES
                 *
02AE  38         U5ERR   SEC                RAISE THE ERROR FLAG
02AF  60                 RTS                AND RETURN TO MONITOR
02B0  EA                 NOP                SO ABOVE CAN BECOME A JMP
                 *
02B1  E0 02      U5STRT  CPX  #2            CHECK FOR 2 PARMS
02B3  F0 22              BEQ  PRMS2         BRANCH IF YES
02B5  B0 F7              BCS  U5ERR         MORE IS TOO MANY PARMS
02B7  E0 01              CPX  #1            HOW ABOUT 1 PARM
02B9  F0 0A              BEQ  PRMS1         BRANCH IF YEP
                 *
02BB  A5 FE              LDA  CURAD         GEE - MUST BE 0 PARMS
02BD  8D 4A A6           STA  P3L           MOVE CURRENT ADDRESS TO P3
02C0  A5 FF              LDA  CURAD+1
02C2  8D 4B A6           STA  P3H           AND FALL THROUGH AS IF 1 PARM
02C5  20 A7 82   PRMS1   JSR  P3SCR         MOVE STARTING ADDRESS TO P.Z.
02C8  18                 CLC                COMPUTE 1 BYTE PAST ENDING ADDRESS
02C9  A5 FE              LDA  CURAD
02CB  69 10              ADC  #16           ***** BYTES PER LINE HERE *********
02CD  8D 4A A6           STA  P3L
02D0  90 0B              BCC  DOOUT         DONE IF NO CARRY
02D2  EE 4B A6           INC  P3H           ELSE TAKE CARE OF CARRY
02D5  D0 06              BNE  DOOUT         AND THEN DONE
02D7  20 9C 82   PRMS2   JSR  P2SCR         2 PARMS HAS STARTING IN P2 - END=P3
02DA  20 93 82           JSR  INCP3         BUMP END BY 1 FOR COMPARE
                 *
02DD  20 16 83   DOOUT   JSR  CRLFSZ        START ON A FRESH LINE
02E0  A2 10              LDX  #16           ***** BYTES PER LINE HERE *********
02E2  A5 FE              LDA  CURAD         SAVE STARTING ADDRESS
02E4  48                 PHA
02E5  A5 FF              LDA  CURAD+1       WILL NEED IT LATER
02E7  48                 PHA
                 *
02E8  20 42 83   ANOTHR  JSR  SPACE         SPACE BETWEEN CHARS
02EB  A0 00              LDY  #0            MAKE SURE REGISTER IS ZERO
02ED  B1 FE              LDA  (CURAD),Y     GET A BYTE OF DATA
02EF  20 FA 82           JSR  OUTBYT        THIS TIME IT IS OUTPUT AS HEX
02F2  20 B2 82           JSR  INCCMP        BUMP TO NEXT BYTE
02F5  B0 03              BCS  LASTPT        DO ASCII PART IF TO END
02F7  CA                 DEX                ELSE COUNT BYTES THIS LINE
02F8  D0 EE              BNE  ANOTHR        DO ANOTHER IF ROOM LEFT
                 *
02FA  20 3F 83   LASTPT  JSR  SPC2          2 SPACES BEFORE ASCII STARTS
02FD  68                 PLA                RESET CURAD BACK TO START
02FE  85 FF              STA  CURAD+1
0300  68                 PLA
0301  85 FE              STA  CURAD
0303  A2 10              LDX  #16           ***** BYTES PER LINE HERE *********
0305  B1 FE      ASCOUT  LDA  (CURAD),Y     GET CHARACTER TO GO AS ASCII
0307  C9 20              CMP  #$20          MAKE SURE NOT CONTROL
0309  90 04              BCC  MAKSPC        MAKE IT SPECIAL IF SO
030B  C9 5B              CMP  #$5B          AS SHOULD DO FOR LOWER CASE
030D  90 02              BCC  OKDO          BRANCH IF NOT SPECIAL
030F  A9 5F      MAKSPC  LDA  #$5F          INSERT FILLER CHARACTER
0311  20 47 8A   OKDO    JSR  OUTCHR        OUTPUT THE ASCII
```

```
0314   20 B2 82     00212           JSR   INCCMP      BUMP TO NEXT BYTE
0317   B0 05        00213           BCS   U5DONE      DONE IF NOW TO THE END
0319   CA           00214           DEX               ELSE NEXT BYTE
031A   D0 E9        00215           BNE   ASCOUT      SANE LINE IF NOT TO END
031C   F0 BF        00216           BEQ   DOOUT       ELSE START A NEW LINE
                    00217 *
031E   18           00218 U5DONE    CLC               CLEAR ERROR FLAG
031F   60           00219           RTS               AND RETURN TO NONITOR
031F                00220 PGMEND    EQU   *-1         END OF PROGRAM ADDRESS NARKER
 LOC   ----OBJECT----  STMT          SYM SYSTEN ADDRESS AND ROUTINES

                    00222 *************************************************************
                    00223 * SYN SYSTNE ROUTINE EMTRY PDINTS AND RAN ADDRESSES          *
                    00224 *************************************************************

8109                00226 INBYTE    EQU   $8109       IMPUT 2 HEX DIGITS INTO 'A'
8293                00227 INCP3     EQU   $8293       INCRENENT P3 BY 1
829C                00228 P2SCR     EQU   $829C       PUT PARM2 INTO 'CURAD'
82A7                00229 P3SCR     EQU   $82A7       PUT PARM3 INTO 'CURAD'
82B2                00230 INCCMP    EQU   $82B2       BUMP 'CURAD' & CONPARE TO PARM3
82BE                00231 DECCMP    EQU   $82BE       SUBTRACT 1 FROM 'CURAD'
82FA                00232 OUTBYT    EQU   $82FA       PRIMT A (2 HEX DIGITS)
8316                00233 CRLFSZ    EQU   $8316       OUTPUT CR/LF AND 'CURAD'
833F                00234 SPC2      EQU   $833F       OUTPUT 2 SPACES
8342                00235 SPACE     EQU   $8342       OUTPUT 1 SPACE
834D                00236 CRLF      EQU   $834D       OUTPUT CR/LF
8972                00237 BEEP      EQU   $8972       TODT THE ONBOARD BEEPER
8A47                00238 OUTCHR    EQU   $8A47       OUTPUT ASCII FROM 'A'
                    00239 *************************************************************
A600                00240 SCPBUF    EQU   $A600       SCOPE OUTPUT BUFFER AREA
A61E                00241 SCPBUD    EQU   $A61E       BYTES USED BIT MAP
A61F                00242 SCPSTL    EQU   $A61F       SEARCH STRING LENGTH
A64A                00243 P3L       EQU   $A64A       INPUT PARAMETER VALUES
A64B                00244 P3H       EQU   $A64B
A64C                00245 P2L       EQU   $A64C
A64D                00246 P2H       EQU   $A64D
A64E                00247 P1L       EQU   $A64E
A64F                00248 P1H       EQU   $A64F
END OF PASS 2-ERRORS=  00000 *********************************************************
```

*Microcomputing is Nick Vrtis' hobby. He is employed by Lear Siegler, Inc. as a Senior System Software Specialist. For this, he works mainly on operating systems on the company's IBM computers, but he also delves into CICS and communication somewhat.*

*His system at home is a SYM-1. It has 5K RAM, soon to be expanded to 8K. He also has Synertek BASIC and has played with Tom Pitman's Tiny Basic, which he has disassembled and modified. His current terminal is an old Datapoint 3300, and he also has a Radio Shack Quick Printer II hooked up through the TTY pot on the SYM. The assemblies that he gets are done with a cross assembler that he wrote to run on the IBM gear.*

# Sorting Revealed

**A truly fresh approach to understanding tha basics of sorting. In addition to a particularly lucid discussion of various sorting mathods, programs are prasanted which damonstrata the sorting algorithms in action.**

**Richard C. Vile, Jr.**

It has often been said that a picture is worth a thousand words. Sadly, this maxim is frequently ignored by professional educators, especially when dealing with such bone-dry subjects as mathematics and computer science. This article will present a detailed example of the use of a simple, yet effective, visual technique for giving insight into the basis for certain algorithms. Our approach will be to show the algorithm in action. Our medium will be the Apple II personal computer, but any computer which provides a memory-mapped display will do. The vehicle for the demonstration will be one of the staples of the computer science curriculum — the joy of pedants and the bane of poor benighted students — viz. sorting algorithms.

**Sorting Theory**

Unfortunately, we must stoop to pedantry to begin with. The reader who is already well-versed in sorting lore may skip directly to Sorting Implemented.

Sorting is such a varied and vast topic that large portions of entire books have been devoted to it. Perhaps the best known compendium of sorting facts and theory is to be found in Knuth's robust volume Sorting and Searching (The Art of Computer Programming Vol. 111, Addison Wesley, 1973). Our demonstration will be limited to just a few of the better known sorting algorithms, although the techniques could be applied to others as well. We shall provide programs that allow the visualization of five different sorting algorithms: bubble sort, Shell sort, insertion sort, selection sort, and quicksort. Of these, we shall discuss the bubble sort and quicksort in some detail prior to the presentation of the programs. Details of the others may be found in almost any good introductory computer science text, as well as in most texts on data structures.

Apart from the specific details of the algorithms used, the theory connected with sorting deals with efficiency. When people who are "in the know" discuss sorting, they will frequently bandy about certain terminology which they don't bother to explain. In hopes of increasing the number of cognoscenti involved in such discussions, we shall now attempt to lay out some of the more common terms for you.

To simplify matters somewhat, let us assume that all of our sorting will take place entirely in memory. Sorting methods that involve storing intermediate stages on disk files or magnetic tape, so-called external sorts, will be beyond our scope, although presumably not beyond our ken. The objects to be sorted will be assumed to be numbers, either integer or floating point, stored in memory in an array of one dimension and of a given size. The size of the array being sorted will be a hit personality throughout the discussion, so we give it a name. It will be denoted by N.

Number of elements to sort = N

In order to fully comprehend one of the definitions to be given later, it is necessary to indulge in a bit of mathematics. We shall need to understand two functions. In particular:

$\mathrm{Log}_2 x$ = base 2 logarithm of x
$\lfloor x \rfloor$ = floor of x

Actually, we are interested in the combination of these functions as applied to the friendly value N:

$\lfloor \log_2 N \rfloor$

i.e. the floor of the base 2 logarithm of N. Before you run screaming to the nearest math anxiety clinic, at least read the next few sentences of explanation.

Suppose you have a pile of N coconuts (why coconuts, you ask? Why not, we reply!). Think about the following process:

1. Subdivide the pile into two piles which are as nearly equal in size as possible.

2. Take the smaller of the two piles from step 1. If it consists of one coconut, then stop. Otherwise, repeat from step 1.

Now how many times did you do step 1? The answer is the value of $\lfloor \log_2 N \rfloor$! So, without worring about picky details, the floor of the base 2 logarithm of N is the number of times you can divide N by 2 and still retain a non-zero quotient. Figure 1. pictures a simple case.

An alternate way of thinking about

the situation involves collecting coconuts. The procedure is as follows:

1. Begin with a single coconut.

2. If doubling the number, k, of coconuts which you already have would cause your total to exceed N coconuts (2k is greater than or equal to N), then stop.
3. Collect k more coconuts, giving you 2k, and repeat step 2 now thinking of the new total as the value of k.

Now how many times did you execute step 3? The answer will again be $[\log_2 N]$. Before you go on, try to convince yourself (without flying to Tahiti to collect real coconuts),the two procedures yield the same result.

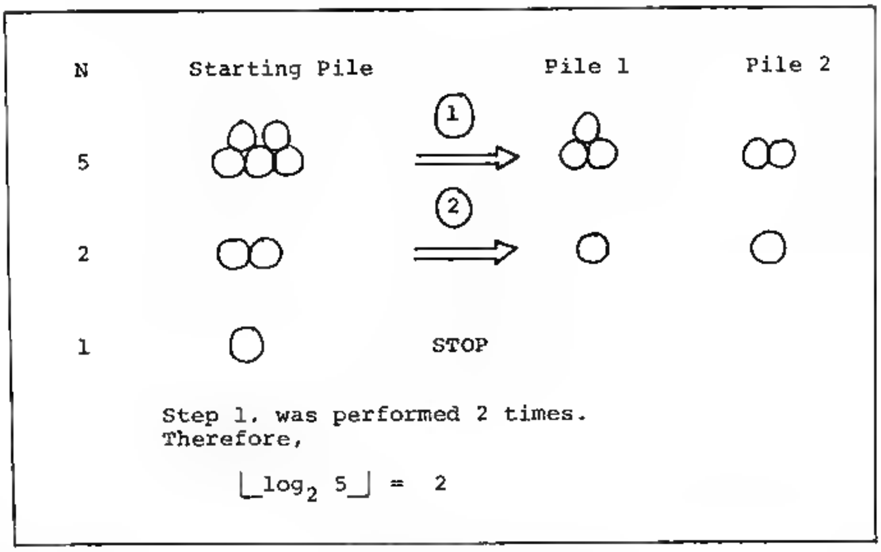



*Figure 1*

We shall return to this value, the "coconut number", later.

In order to talk about the efficiency of any algorithm, we need some quantities that we can measure. For sorting algorithms, we concentrate on two: the number of comparisons and the number of interchanges.

A comparison occurs whenever a member of the collection of numbers is compared to something else. The something else could be a value fished out of a hat, or it could be another member of the collection. Thus, a statement such as IF A(I)>A(I+1)THEN...counts as a comparison, as well as IF A(I)> MAX THEN...

An interchange occurs whenever a member of the collection of numbers is moved from one place to another in the computer's memory, and possibly some other number takes its place. The classic interchange may be described by the sequence of three statements:

```
TEMP = A(I)
A(I) = A(J)
A(J) = TEMP
```

(assuming, ofcourse, that $I \neq J$). Not all sorting algorithms use this classic form, but there is usually an easily identified interchange step whose repetition we can count.

Trying to count the number of comparisons and/or interchanges which take place during the course of execution of a sorting algorithm will give an approach to measuring the efficiency of that algorithm. In addition to comparisons and interchanges, there will also be overhead involved in a sorting algorithm: i.e. the computing time used in loop control, recursion, etc. This is more difficult to measure theoretically and is therefore usually deduced from empirical observations.

Being armed with a few terminological weapons, we may now attack some of the more familiar sorting buzz phrases. Assume we are speaking of the number of comparisons made during the execution of some sorting algorithm. Then we may speak of an $N^2$ sorting algorithm (pronounced N-squared). This means that "on the order of" N times N comparisons will be made in the course of sorting an array of size N. Well, that was relatively painless — at least as a definition! The interesting (painful) part comes when we try to prove that a given algorithm is an $N^2$ algorithm. We shall get to that in the next section.

Another phrase which is frequently encountered when casually "talking sorts" is: that's an N log N sort (pronounced N log N!). What that actually means is that the expected number of comparisons in carrying out the sorting algorithm for an array of size N is:

$$N * ([\log_2 N])$$

That is, N multiplied by the coconut number. Again, this is easy enough to say, but perhaps a bit harder to appreciate than the $N^2$ description. After all, why should we be concerned with these numbers, and what is the significance of the difference between them?

Consider briefly, Table 1. It shows values for N, $N^2$, $[\log_2 N]$, and $N * [\log_2 N]$.Assuming that overhead is relatively constant, or at least negligible from one algorithm to the next, we see that there is an ever increasing difference between $N^2$ and NlogN (from now on, we assume that logN means $[\log_2 N]$). To make the comparison more concrete, let us assume that a comparison costs .001¢, and that we need to sort an array containing 1,048,576 numbers. Using an $N^2$ sort will cost USD 10,995,116.27, whereas using an NlogN sort will only put us out USD 209.72 Of course, a single comparison of two numbers on today's monster computers or "big iron" as they are sometimes referred to in the trade- costs considerably less than. .001¢. But even at .0000001¢ per comparison - a rate of 10,000,000 comparisons per penny- the cost differential will be 2¢ for the NlogN sort-USD 1,099.51 for the $N^2$ sort! With that kind of comparison, you can see why no commercially viable sorting package is going to use the $N^2$ sorting approach.

### Some Sorting Algorithms

We now present two of the more well known sorting algorithms in some detail. We will attempt informally to prove that the first is an $N^2$

algorithm. The second algorithm discussed is an example of an NlogN algorithm, but we shall spare the reader any attempts at proof.

### Bubble Sort

This algorithm is probably the most widely known and loathed by students of introductory computer science. Many an instructor has droned on about its properties to unwilling students of FORTRAN! For many of these students, it is their only taste of the vast menu of sorting techiniques.

We assume that N elements, which we shall denote by A(1), A(2),..., A(N), are to be arranged in ascending order; in short, sorted. The bubble sort operates by making repeated "sweeps" through the array, causing various elements to "bubble — up" in the process. We shall see that for each sweep, at least one element is guaranteed to be positioned in its correct final slot in the array.

The heart of each sweep is the idea of comparing two adjacent entries in the array:

A(I) A(I + 1)

If A (I) has a greater value than A(I + 1), then the two elements are known to be out of correct order and need to be swapped. This is accomplished by the use of the classic interchange, which we illustrate here in BASIC and Pascal:

BASIC

```
100     IF A(I) <=  A(I+1) THEN 140
110       TEMP = A(I)
120       A(I) = A(I+1)
130       A(I+1) = TEMP
140     ...
```

Pascal

```
if  A[I] >  A[I+1] then
begin

  Temp := A[I];
  A[I] := A[I+1];
  A[I+1] := Temp;

end;
```

***Figure 2***
The "Classic Interchange"

| N | $N^2$ | log N | N log N |
|---|---|---|---|
| 64 | 4096 | 6 | 384 |
| 128 | 16,384 | 7 | 896 |
| 256 | 65,536 | 8 | 2,048 |
| 512 | 262,144 | 9 | 4,608 |
| 1,024 | 1,048,576 | 10 | 10,240 |
| 2,048 | 4,194,304 | 11 | 22,528 |
| 4,096 | 16,777,216 | 12 | 49,152 |
| 8,192 | 67,108,864 | 13 | 106,496 |
| 16,384 | 268,435,456 | 14 | 229,376 |
| 32,768 | 1,073,741,824 | 15 | 491,520 |
| 65,536 | 4,294,967,296 | 16 | 1,048,576 |
| 131,072 | 17,179,869,184 | 17 | 2,228,224 |
| 262,144 | 68,719,476,736 | 18 | 4,718,592 |
| 524,288 | 274,877,906,944 | 19 | 9,961,472 |
| 1,048,576 | 1,099,511,626,776 | 20 | 20,971,520 |

***Table 1***

Now consider the iterations of this fundamental step which are necessary in order to bring the entire array into sorted order. First, suppose we are just beginning. Then we can make no assumptions about the sizes of the array elements, relative to their positions in the array. Thus, suppose we iterate the fundamental compare-maybe-swap step over values of I ranging from 1 to N-1 (why not 1 to N?). That is, we will successively compare A(1) and A(2),A(2) and A(3), and so on, until we reach A (N-1) and A(N). Positions of various elements will change through swapping. In particular, the largest numerical value in the orignal array is guaranteed to wind up in A(N). Positions of various elements will change through swapping. In particular, the largest numerical value in the original array is guaranteed to wind up in A(N) after the sweep is completed. To convince yourself that this is true, ask;"If the largest value is originally in A(J), then what other array entries will it be swapped with?"

The last paragraph has indicated that we can reach a picture such as that shown in Figure 3, after one sweep of the array. What has been accomplished? We have partially sorted the original array. How much of the resulting array is now in correct order? One element — the last. Note that this is the same as the number of sweeps we have made. Now suppose we make a second sweep through the array, comparing A(1) and A(2), A(2) and A(3), etc. until we reach A(N- 2) and A(N-1). It is not necessary to compare A(N-1) and A(N), since we know that A(N) is already in its correct final position. Moreover, A(N-1) is now also guaranteed to be the second largest element in the array, and therefore in its correct final position. Thus the original array has been divided into two pieces: the elements A(1), A(2), ... A(N-2), still possibly unsorted, and the elements A(N-1) and A(N), both where they 'should be'. We have made two passes and put two elements in their correct positions.

Continuing this process by making passes through less and less of the array will cause more and more of the 'tail end' of the array to be in correct final order and leave less and less of the beginning of the array to still be sorted. Altogether it will take N-1 passes through the array to guarantee that it is totally sorted. The reason that it does not require N passes is that the last pass causes two elements to wind

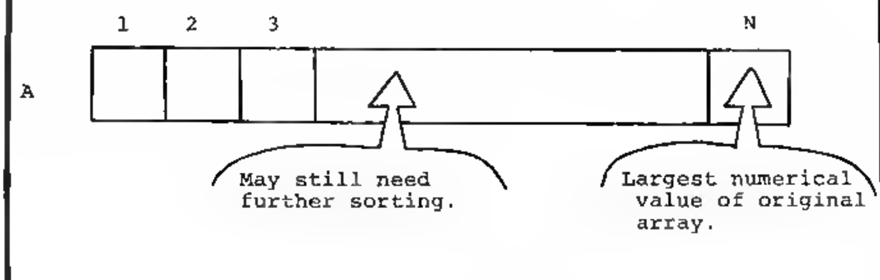



*Figure 3*
Array after sweep of Bubblesort

up in their correct places, instead of just one. Figure 4 gives both a BASIC and a Pascal version of the complete bubble sort algorithm.

Now let us see if we can count the number of comparisons that will be made. Each sweep through the array corresponds to one pass through the inner loop of the algorithm. The number of comparisons made will be the same as the value of the upper limit of this loop, which according to Figure 4. Is N-I. The value of I is varried by the outer loop and runs from 1 to N-1. Thus, there will be:
N-1 comparisons the first time through the loop.
N-2 comparisons the second time through the loop.
N-3 comparisons the third time through the loop.
... ... ... ... ... ...
N-(N-2) = *2* comparisons the (N-2)nd time through the loop
N-(N-1) = *1* comparisons the (N-1)st time through the loop.
The total number is therefore:
(N-1) + (N-2) + ... + 3 + 2 + 1
This number is known in mathematics as a 'triangular' number, and by a formula from algebra may be expressed solely in terms of N as 1/2 ($N^2$ - N). Consequently, there are about $N^2$ comparisons made.

The inefficiency of the bubble sort is compensated for by its simplicity, especially from a pedagogical point of view. It is totally trivial to program, as we have seen. Consequently, it is quite acceptable for sorting tasks that only involve 'small' values of N.

**Quicksort**

Quicksort, invented by C.A.R. Hoare, is probably the most 'elegant' of the sorting techniques yet devised. It is an NlogN sort, which is based on a very simple idea and in its most compact form may be programmed in very few lines of code. In fact, probably the greatest difficulty in grasping how it works involves understanding the administrative details of how to apply the basic step which motivates its operation. One has the tendency to say, 'You mean, that's all there is to it?', or 'But what do you mean by simply apply the same procedure to both halves?'. Nonetheless, once appreciated, it is an algorithm you will never forget. That should be reward enough for the effort expended in understanding it in the first place.

The basic idea underlying Quicksort is to perform interchanges of non-adjacent array elements in hopes of bringing order to the array more quickly (bubble sort has already demonstrated the inefficiency of interchanging adjacent entries). The idea is applied using the concept of a *partition* of the array elements.

To partition the elements A(P), A(P+1), ..., A(Q) of the array A, where $P \geq 1, P \leq Q, Q \leq N$, requires that some value X which actually occurs as one of the entries A(P), A(P+1),...,A(Q) be placed into its correct final position, say K, and that the remaining elements are arranged so that $A(I) \leq A(K)$ for O<K and $A(J) \geq A(K)$ for J>K. The results are pictured in Figure 5.

For convenience in implementation (although this may not be the optimal choice in theory), we shall always choose A(P) as the value X, which is to be inserted into its correct final resting place. To accomplish our end result, we adopt the following 'double-barreled' scan:

Start with I=P+1 and J=Q. Scan forward from I (i.e. in increasing I-value order) until we find A(I) for which $A(I) \geq X$. Scan backward from J (i.e. in decreasing J-value order) until we find A(J) for which $A(J) \leq X$. Then interchange A(I) and A(J), since they are both in the 'wrong half' of the partition according to the above definition. Continue this procedure until $J \leq I$. As a final act, interchange A(P) and A(I), where I now has its 'final' value. This puts X=A(P) into its correct final position in the array. You should convince yourself that it also achieves the picture shown in Figure 5. Actually, there is one case which fails. See if you can discern what it is — we'll come back to it later on.

An example may make things a bit clearer. Figure 6 shows an un-

BASIC

```
10    FOR I = 1 TO N-1
20    FOR J = 1 TO N-I
30    IF A(J) <= A(J+1) THEN 70
40    TEMP = A(J)
50    A(J) = A(J+1)
60    A(J+1) = TEMP
70    NEXT J
80    NEXT I
```

Pascal

```
for I := 1 to N-1 do
  for J := 1 to N-I do
    if A[J] > A[J+1] then
    begin

      Temp := A[J];
      A[J] := A[J+1];
      A[J+1] := A[J];
    end;
```

*Figure 4*
Bubble sort algorithm in both BASIC and Pascal

sorted array of 16 elements, which is to be partitioned for P = 1, Q = 16. Shown are the first values of I and J for which an interchange of the partitioning process will take place. See if you can draw the final picture: showing the array with the partition complete and the value of K. The answer is shown in Figure 7.

When one gets down to programming the partitioning process, several details that may not have been previously obvious suddenly force themselves into the spotlight. In order to highlight these, we present in Figure 8 a Pascal procedure for the partition step. The first item which may catch your eye is that array A is indicated in the parameter list to be of size N + 1, instead of N. The reason may be seen by studying the second repeat statement of Figure 8:

```
                                repeat
     I := I + 1
    until A(I) ≥ Value;
```

As with all loops, the programmer should be sure that there is a way out! In this case, if the elements A(1), A(2), ... , A(N) of the array are assumed to be randomly distributed

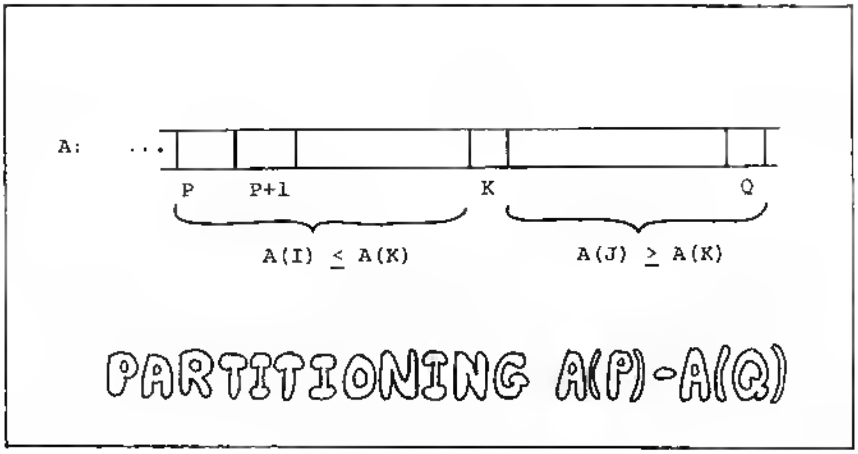



*Figure 5*

among all possible values, then there is no guarantee that any of them satisfies the condition A(I) ≥ Value. Thus, we have extended the array and stored a value in A(N + 1) which is guaranteed to be greater than or equal to any other value that could occur in the original array. In Pascal, the predefined identifier Maxint serves the purpose, and we may assume that the assignment A[N + 1] := Maxint; has occurred in the calling routine. Now, even if all elements of A are strictly less than A(1), the repeat loop will terminate when it bumps into the Maxint value stored in A[N + 1]. Such a value, which is not part of the data being manipulated, but instead serves to protect against some dire circumstances, is known as a *sentinel*.

This approach raises two further questions: first, do we face a similar problem with J; and second, do we face the possibility of erroneously swapping A(N + 1) with some element of A. The first question is easily answered by realizing that Value := A [Lower]. Thus, if J is decreased so far that J := Lower, then A[J] ≤ Value is automatically true. Thus, the first repeat loop is guaranteed to stop because of this choice. To answer the second question, let's look closely at what happens when N = Upper and A(I) < Value for all I, I = 2,3, ... ,N. The repeat statement:

```
repeat
      J := J — 1
    until A[J] ≤ Value
```

Immediately succeeds. J starts at N + 1, J—1 = N and A(N) < Value by our assumption. Thus, J stops at the value N after the first time through the loop. On the other hand, the repeat statement for I will continue to fail, again by our assumption, until I = N + 1. Now I + N + 1 and J = N. This means that the test I < J will fail. Therefore, the interchange shown inside the while loop will be skipped. Aha!, you say — caught you · nothing happens and Quicksort is a sham!! Fortunately, that is not true. The last two statements in the procedure:

```
      A[Lower] := A[J];
      A[J] :=  Value;
```

will be carried out, causing A[Lower] and A[N] to be swapped.

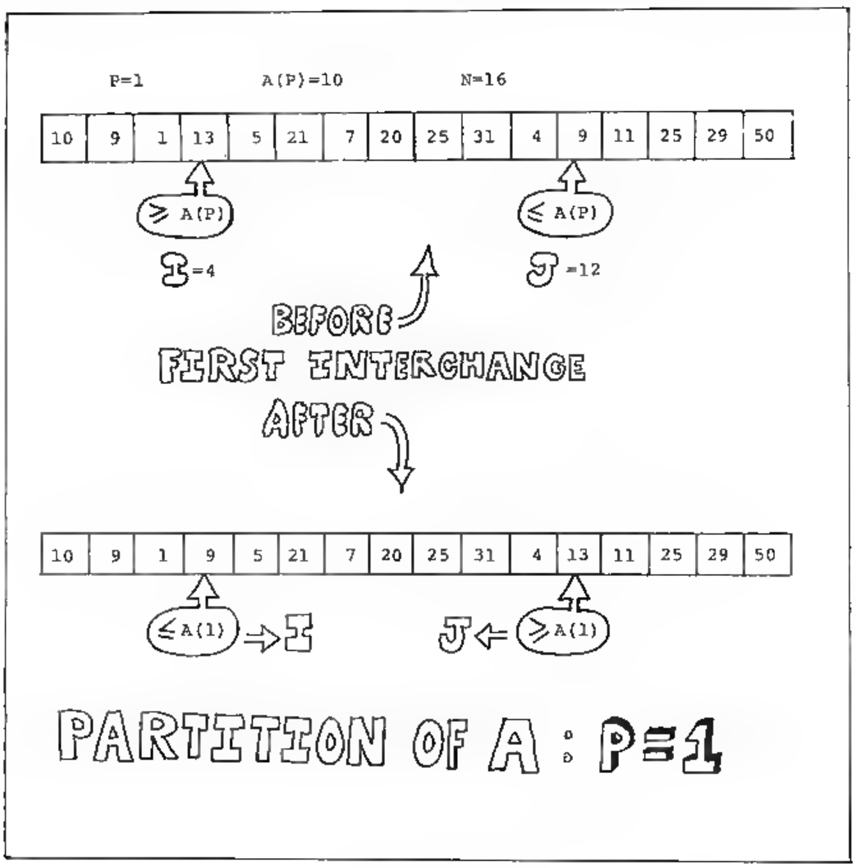



*Figure 6*

To assimilate the code of the pro-

cedure, simulate its action on the array of Figure 6. As a final note, the procedure protects itself from funny initial values for Lower and Upper, by first checking to make sure that Lower < Upper. This will turn out to be necessary in one version (the recursive one) of the complete Quicksort algorithm, but must be moved back to the caller for the other version (the 'straight' or iterative one).

Now that we have studied the innards of the Quicksort algorithm, it is time to investigate how the partition step fits into the larger scheme of things. Once the original array A has been partitioned, we are left with one element in its correct final resting place and two subarrays that remain to be sorted. The beauty of Quicksort is that that is all that remains to be done. Once the two subarrays are both sorted, the entire array is automatically sorted. This is true because of the condition — guaranteed by the partition step — that all elements in the first half of the array arre less than or equal to all the elements in the second half of the array. Not convinced? Think about it! Or, consider the following analogy: a school teacher wishes to arrange test papers in alphabetical order. The papers are divided into two piles (partitioning step) with all papers in the left-hand pile belonging to students whose names begin with letters A to M, and all papers in the right-hand pile belonging to students with names beginning with letters N to Z. Now, if the left-hand pile is arranged (by whatever method) into alphabetical order and likewise the right-hand pile, then all that remains to put the whole collection into alphabetical order is to place the left-hand pile on top of the right-hand pile.

To continue the Quicksort algorithm, one applies the basic step to both subarrays obtained from the first partitioning step. That will produce in each case two new subarrays (or better, sub-subarrays), to which the partitioning process is applied in turn. Since we started with a finite number of elements in array A, sooner or later this will produce sub-sub...subarrays with 0 elements. Such subarrays are sorted by default. Thus, they need not be partitioned any further. Morever, when both subarrays of a

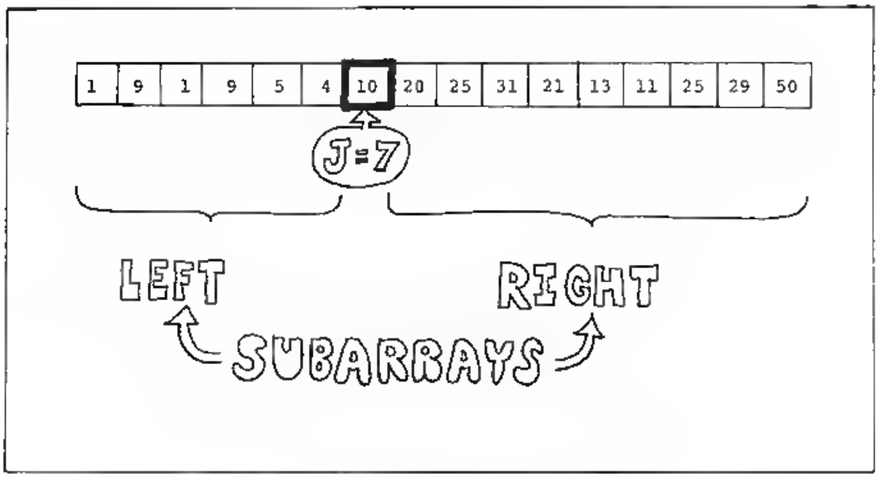



***Figure 7***
Partition step complete A(7) in correct position.

```
procedure

  Partition(

    var  A: array[1..N+1] of integer;
         Lower, Upper:       integer;
    var  J:                  integer );

  var

    Value,Temp:    integer;

  begin

    if Lower < Upper then begin

      I := Lower;    {Lower bound in A for partition step}
      J := Upper;    {Upper bound in A for partition step}
      Value := A(Lower);    {Comparison value for partitioning}

      while I < J do begin  {Partitioning loop}

        repeat       {Find element in right half to switch}
          J := J-1
        until  A(J) <= Value;

        repeat       {Find element in left half to switch}
          I := I+1
        until  A(I) >= Value;

        if I <= J then begin    {Perform the switch}

          Temp := A(J);
          A(J)  := A(I);
          A(I)  := Temp

        end  {of if I <= J}

      end  {of while I < J}

      A(Lower) := A(J);       {Insert A(Lower) into its   }
      A(J) := Value;          {correct final position in A}

    end  {of if Lower < Upper}

  end  {of Procedure Partition};
```

***Figure 8***

```
procedure

  Sort(

    var A: array[1..N+1] of integer;
        Lower,Upper:        integer );

  var

    J:    integer;

  begin

    Partition(A,Lower,Upper,J);   {Partition A between       }
                                  {A(Lower) and A(Upper)     }
    Sort(A,Lower,J-1);            {Sort the "left" subarray  }
    Sort(A,J+1,Upper);            {Sort the "right" subarray }

  end  {of Procedure Sort};
```

***Figure 9***

given subarray reach this state, they form together with their partition element a sorted subarray, which may then be ignored while the remaining unsorted subarrays are processed. Eventually, the original two subarrays will have been sorted and voila!, A will have been sorted. Figure 9 shows the implementation of this scheme as a Pascal procedure must be invoked from outside itself with initial values for Lower and Upper, which are presumably 1 and N, in most cases. Once it gets going, it calls itself on behalf of the subarrays, and the sub-subarrays, etc. until it completely sorts A. Figure 10 shows the progress of the sort as applied to a small array, with N=8. Study it carefully. Figure 11 presents the calling structure to Sort for the array in figure 10. The root of the tree represents the original call to Sort from outside. The interior nodes of the tree represent calls to Sort from within itself. Each node is labeled with the values of Lower and Upper which were passed on the corresponding call. The leaves of the tree represent calls to Sort in which the passed values of Lower and Upper correspond to subarrays with 0 elements. Such subarrays are already sorted and "nothing " will happen on these calls.

EXERCISE: Determine whether or not the Partition procedure may be modified to return whenever the passed array has either 0 or 1 elements. If so, make the necessary changes to the code.

The recursive implementation of Quicksort is without a doubt one of the most "beautiful" algorithms yet devised in any branch of computer science. Unfortunately, the performance of Quicksort in such an implementation, even though superior to most $N^2$ algorithims, is still not quite as good as it could be. We shall not attempt to explain the technical reasons for this, other than to say that recursion involves more than a modicum of overhead. However, we shall attempt to formulate the algorithm in a non-recursive or iterative fashion for comparison.

Now look back at the recursive implementation of Quicksort shown in Figure 9. Since Sort calls itself, this means that the variable J, which is used locally within Sort, must be given a different "incarnation" on each call. Otherwise, the recursive calls would cause its former value to be lost, which in turn would mean that the procedure would get mixed up about where the subarrays began and ended. In languages, such as Pascal, which support recursive procedures, the uniqueness of J on each call is guaranteed. In a language like BASIC, there aren't even procedures, let alone recursive ones! Thus, in such a language, we must "fake it" in some way or another.

What is it about the variable J that's so important? It remembers the dividing point between the two subarrays determined by any partition step. This enables the two halves to be sorted separately by sucessive calls to Sort. Another way to approach matters would be to save information about subarrays that still need sorting and retrieve it as necessary. An appropriate data structure for preserving such information is a stack. The Lower and Upper values for one "half" of a partition may be saved by pushing them onto the stack, while the other "half" is being sorted. When the other half has been completely sorted, the Lower and Upper values for the saved half may be popped off the stack and the sorting of that half commenced. Of course while sorting a given half, new pairs of bounds for smaller subarrays will be determined and bounds for one subarray of each such pair will in turn be pushed onto the stack. If a point is reached at which we try to pop the bounds of a subarray from the stack, and find that the stack is empty, then we will know that the original array is completely sorted. As a performance enhancement, we shall always sort the smaller of any given pair of subarrays first. This is in distinction to the algorithm of Figure 9, which always sorts the left subarray first. Sorting the smaller subarray first will cause a minimum number of entries to be saved on the stack.

The actual code of an iterative implementation of the Quicksort algorithm is presented in Listing 5, using APPLE Integer BASIC.

**Sorting Implemented**

The APPLE II Integer BASIC programs of Listings 1-5 provide implementations of visual sorts for the following five methods: Bubble sort, straight insertion sort, selection sort, Shell sort, and Quicksort. The visual display arranges the array to be sorted as a table of up to 100 positive two digit integers — the user may request fewer if so desired to speed up the completion of the algorithm. The basic table using the random number generator for INTEGER BASIC. For skeptical viewers, the values 0 to N may be generated in a permuted order and filled into the first N+1 slots of the tableau. The modification needed in order to accomplish this is shown in Figure 12. Figure 13 shows a typical tableau, this one prior to the beginning of Shellsort. Notice that extra information is displayed in the small area surrounding the display. By studing the listing and carefully

| | | | | A | | | | Call |
|---|---|---|---|---|---|---|---|---|
| 10 | 9 | 1 | 13 | 5 | 21 | 7 | 20 | Partition(A,1,8); |
| 10 | 9 | 1 | 7 | 5 | 21 | 13 | 20 | |
| 5 | 9 | 1 | 7 | $\boxed{10}$ | 21 | 13 | 20 | |
| 5 | 9 | 1 | 7 | $\boxed{10}$ | 21 | 13 | 20 | Partition(A,1,4); |
| 5 | 1 | 9 | 7 | $\boxed{10}$ | 21 | 13 | 20 | |
| 1 | $\boxed{5}$ | 9 | 7 | $\boxed{10}$ | 21 | 13 | 20 | |
| 1 | $\boxed{5}$ | 9 | 7 | $\boxed{10}$ | 21 | 13 | 20 | Partition(A,1,1); |
| $\boxed{1}$ | $\boxed{5}$ | 9 | 7 | $\boxed{10}$ | 21 | 13 | 20 | |
| $\boxed{1}$ | $\boxed{5}$ | 9 | 7 | $\boxed{10}$ | 21 | 13 | 20 | Partition(A,3,4); |
| $\boxed{1}$ | $\boxed{5}$ | 7 | $\boxed{9}$ | $\boxed{10}$ | 21 | 13 | 20 | |
| $\boxed{1}$ | $\boxed{5}$ | 7 | $\boxed{9}$ | $\boxed{10}$ | 21 | 13 | 20 | Partition(A,3,3); |
| $\boxed{1}$ | $\boxed{5}$ | $\boxed{7}$ | $\boxed{9}$ | $\boxed{10}$ | 21 | 13 | 20 | Partition(A,5,4); |
| $\boxed{1}$ | $\boxed{5}$ | $\boxed{7}$ | $\boxed{9}$ | $\boxed{10}$ | 21 | 13 | 20 | Partition(A,6,8); |
| $\boxed{1}$ | $\boxed{5}$ | $\boxed{7}$ | $\boxed{9}$ | $\boxed{10}$ | 20 | 13 | $\boxed{21}$ | |
| $\boxed{1}$ | $\boxed{5}$ | $\boxed{7}$ | $\boxed{9}$ | $\boxed{10}$ | 20 | 13 | $\boxed{21}$ | Partition(A,6,7); |
| $\boxed{1}$ | $\boxed{5}$ | $\boxed{7}$ | $\boxed{9}$ | $\boxed{10}$ | 13 | $\boxed{20}$ | $\boxed{21}$ | |
| $\boxed{1}$ | $\boxed{5}$ | $\boxed{7}$ | $\boxed{9}$ | $\boxed{10}$ | 13 | $\boxed{20}$ | $\boxed{21}$ | Partition(A,6,6); |
| $\boxed{1}$ | $\boxed{5}$ | $\boxed{7}$ | $\boxed{9}$ | $\boxed{10}$ | $\boxed{13}$ | $\boxed{20}$ | $\boxed{21}$ | Partition(A,8,7); |
| $\boxed{1}$ | $\boxed{5}$ | $\boxed{7}$ | $\boxed{9}$ | $\boxed{10}$ | $\boxed{13}$ | $\boxed{20}$ | $\boxed{21}$ | Partition(A,9,8); |

***Figure 10***
Complete trace of Quicksort for N = 8 boxed entries are known to be in the correct slot.

monitoring this information, extra insight into the nature of the algorithms may be gained.

All values generated are positive and less than 100. This is done because of horizontal space constraints in the display and does not reflect any inherent limitations in the algorithms themselves.

The programs each carry out one of the sorting algorithms.As the array is sorted, the values displayed on the screen are modified to reflect the changes taking place internally. Various devices are used to highlight this: some visual and some aural. The audio effects are programmed using the Programmer's Aid ROM. Thus, you may have to remove or modify certain statements in order to run the programs, if you don't own PA.

Each time a number is moved from one place to another in the array, that value is highlighted in the display. This is accomplished by momentarily displaying the value in reverse video, then switching back to normal mode. If your APPLE has been modified for lower case, this probably won't work. You can get a good idea of how each algorithm does its job just by watching the pattern of flashes on the screen.* In addition to this, as mentioned above, each sort prints on the border of the display some additional imformation about what is happening. Each program begins with a prologue giving the name of the sort and prompting the user for the number of elements to be sorted. The value of PDL(1) is used by the programs to control the speed at which the display is generated. Thus to slow down the progress of the program, simply turn up the PDL(1) control.

While each algorithm is in progress, two tones will be sounded periodically. One tone is generated each time an array element is copied from one place to another, that is, for each interchange. A different tone is sounded whenever an array element is compared to another or to a fixed value, that is, for each comparison. Listening to the pattern of sounds thus produced gives a very definite auditory tattoo to each algorithm. The calls to Programmer's Aid which produce these tones are localized in subroutines to facilitate their removal or replacement should you not have the PA ROM. For example, in the bubble sort demo, you may defeat the sounds by inserting the two statements:

```
901 RETURN
951 RETURN
```

Even if you do have PA, you may want to use these statements in order to (a) speed up the program a little or (b) hear only comparisons or only interchanges.

*NOTE: If you stop the program with a Control-c at just the right (or wrong — depending on your point of view) moment, you may find that everything is being displayed in reverse video. To return to normal display mode, simply type:

```
POKE 50,255
```

and all should be well.

I hope that these demonstrations will enhance your understanding and enjoyment of sorting algorithms you may wish to implement similar demos for other sorting algorithms, or if you are very ambitious, how about a way of having the various algorithms swap in and out while the same array is sorted in stages? Happy viewing!

A complete package of twenty demonstration programs, including the ones listed here and variations upon them may be obtained for $14.95 on a single diskette by writing to the author.

*3467 Yellowstone Drive*
*Ann Arbor, MI 48105*

μ

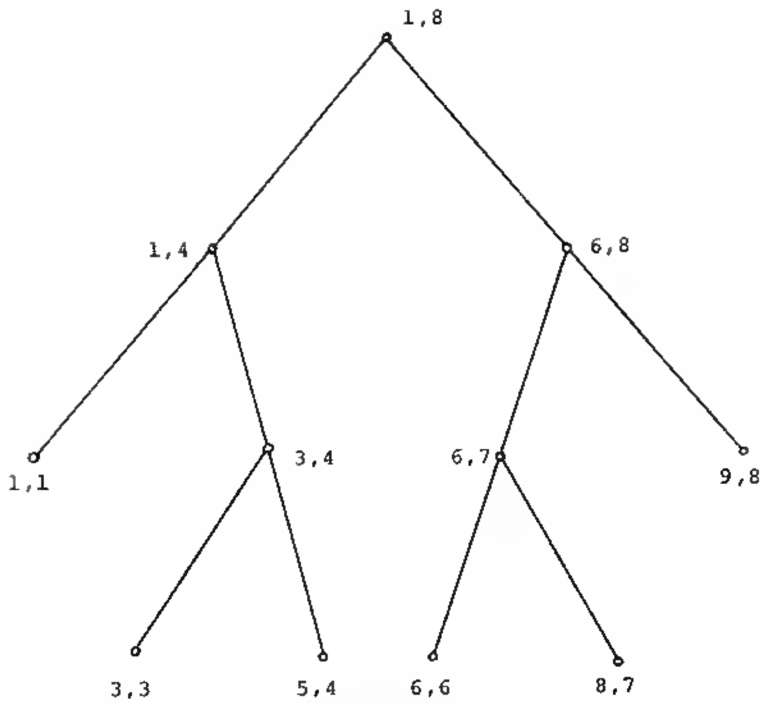



***Figure 11***
Call tree for Figure 10. Each node is labelled with the values of Lower, Upper for the corresponding call. The levels of the tree correspond to the depth of the recursion.

```
     Ø  1  2  3  4  5  6  7  8  9
    ----------------------------
 Ø! 12 72 14 68 54 23 32  3 56 24

 1! 44 26 41  Ø 87 67  8 81 39 39

 2!  3 26 60 64 35 2Ø 39 78 65 26

 3! 16 17 99 69 81 88 65 32  5 68

 4! 37 44 32 89 65 37 2Ø 38 84 77

 5!

 6!

 7!

 8!

 9!

          SHELL SORT       J= 1Ø
           SPAN = 10     A(J)=44
```

***Figure 13***
Just before the start of the shell sort. Fifty elements are being sorted.

```
80   For I =0 TO N: A(I) =: NEXT I
90   For I =0 TO N
100  L = RND (N + L): IF A(L)
                    > =0 THEN 100
105  A(L) = I: X=L: GOSUB
                          DISPLAY
110  NEXT I
```

***Figure 12***
Modification to Display generation: will seed the initial array with exactly the numbers 0 to N in some permuted order.

*Richard Vile was educated in mathematics, earning a B.S. degree from Michigan State University and a Ph.D. from Cornell University.*

*Richard taught mathematics at Eastern Michigan University from 1970 - 1977. While at Eastern, he became interested in computers and began studying and teaching computer science.*

*In early 1978, he took a leave of absence from, E.M.U. in order to work for SYCOR, Inc. and Ann Arbor manufacturer of distributed data processing computer systems. He enjoyed the work so much that he did not return to the academic world. He is currently employed by the same company, known as Northern Telecom Systems Corporation, where he is engaged in the development of languages and language related software: compilers, assemblers, linkage editors, etc.*

*Richard owns an APPLE II computer, which he puts to good use preparing articles for MICRO and other personal computing journals.*

Richard C. Vile, Jr
3467 Yellowstone Dr.
Ann Arbor, Michigan 48105

***Continued on page 24...***

'Sorting' continued from page 22

*Listing 1*
**BUBBLE SORT**

```
>PR#0
>LIST
  5 DIM A(100)
  6 KBD=-16384:CLR=-16368:TITLE=
    500:INTRO=1000
  7 DISPLAY=600:WAIT=800:COMPARE=
    900:INTERCHANGE=950
  8 MUSIC=-10473:TIME=766:TIMBRE=
    765:PITCH=767
 10 TEXT : CALL -936
 20 GOSUB INTRO
 50 GOSUB TITLE
 90 FOR R=0 TO 100:A(R)=32767: NEXT
    R
100 FOR I=0 TO N
105 A(I)= RND (100):X=I: GOSUB
    DISPLAY
108 IF N=0 THEN 150
110 NEXT I
150 FOR I=1 TO NUM-1
152 FLAG=0
155 FOR J=0 TO N-I
158 FOR T=0 TO PDL (1): NEXT T
159 GOSUB COMPARE
160 IF A(J)<=A(J+1) THEN 200
163 X=100: POKE 50,127:A(100)=A(
    J): GOSUB DISPLAY
165 KEEP=A(J): GOSUB INTERCHANGE:
    X=J
170 POKE 50,63
173 A(J)=A(J+1): GOSUB DISPLAY:
     GOSUB INTERCHANGE: POKE 50
    ,255
175 GOSUB DISPLAY:X=J+1: POKE 50
    ,63
180 A(J+1)=KEEP: GOSUB DISPLAY:
     GOSUB INTERCHANGE: POKE 50
    ,255
185 GOSUB DISPLAY
190 FLAG=1
195 KEY= PEEK (KBD): IF KEY<128
     THEN 200
196 POKE CLR,0: GOSUB WAIT
200 NEXT J
202 IF FLAG=0 THEN 208
205 NEXT I
208 VTAB 24: TAB 21: PRINT "FINISHE
    ";
210 IF PEEK (KBD)<128 THEN 210
220 POKE CLR,0: CALL -936: GOTO
    20
500 TEXT : CALL -936
510 VTAB 1: FOR I=0 TO 9: TAB 7
    +3*I: PRINT I;: NEXT I
515 VTAB 2: TAB 7: FOR I=0 TO 9
    : PRINT "---";: NEXT I
520 FOR J=0 TO 9: VTAB 3+2*J: TAB
    4: PRINT J;"! ";: NEXT J
525 VTAB 23: TAB 1: PRINT "TEMP="
    ;: TAB 20

528 PRINT "BUBBLE SORT"
530 RETURN
600 COL=X MOD 10
610 ROW=X/10
620 VTAB 2*ROW+3: TAB 7+3*COL
630 IF A(X)<10 THEN PRINT " ";
635 PRINT A(X);
640 RETURN
800 IF KEY<> ASC("Q") THEN 810
805 TEXT : CALL -936: END
810 VTAB 2*ROW+3: TAB 6+3*COL: PRINT
    ">";
815 KEY= PEEK (KBD): IF KEY<128
     THEN 810
817 VTAB 2*ROW+3: TAB 6+3*COL: PRINT
    " ";
820 POKE CLR,0: RETURN
900 REM  *** TO REMOVE SOUND FOR COM
    PARISONS - INSERT 901 RETURN ***

902 POKE PITCH,10: POKE TIME,5:
     CALL MUSIC
905 FOR DE=1 TO PDL (1): NEXT DE
910 RETURN
950 REM  *** TO REMOVE SOUND FOR INT
    ERCHANGES - INSERT 951 RETURN **
    *
952 POKE PITCH,49: POKE TIME,3:
     CALL MUSIC
955 FOR DE=1 TO PDL (1): NEXT DE
960 RETURN
1000 VTAB 10: TAB 5: PRINT "I WILL SO
    RT UP TO 100 POSITIVE"
1001 TAB 5: PRINT "INTEGERS INTO ASCE
    NDING"
1002 TAB 5: PRINT "ORDER USING THE BU
    BBLE SORT."
1008 VTAB 15: TAB 10: INPUT "VALUE OF
     N PLEASE",NUM:N=NUM-1
1010 IF NUM<=100 THEN RETURN
1015 TAB 10
1020 PRINT "TOO BIG!!!!!": GOTO
    1000
```

***Listing 2***
**INSERTION SORT**

```
>PR#0
>LIST
  0 I=J=Y=N
  5 DIM A(99)
  6 KBD=-16384:CLR=-16368:TITLE=
    500:INTRO=1000
  7 DISPLAY=600:WAIT=800:COMPARE=
    900:INTERCHANGE=950
  8 MUSIC=-10473:TIME=766:TIMBRE=
    765:PITCH=767
  9 DELAY=975:ERASE=650
 10 TEXT : CALL -936
 20 GOSUB INTRO
 50 GOSUB TITLE
 90 FOR R=0 TO 99:A(R)=32767: NEXT
    R
100 FOR I=0 TO N
105 A(I)= RND (100):X=I: GOSUB
    DISPLAY
108 IF N=0 THEN 150
110 NEXT I
150 FOR I=1 TO N
151 IF I>N THEN 206:Y=A(I)
152 VTAB 23: TAB 32: PRINT "I="
    ;: IF I<10 THEN PRINT " ";:
     PRINT I
153 VTAB 24: TAB 32: PRINT "Y="
    ;: IF Y<10 THEN PRINT " ";:
     PRINT Y;
154 GOSUB INTERCHANGE
155 FOR J=I-1 TO 0 STEP -1
156 GOSUB DELAY:KEY= PEEK (KBD)
    : IF KEY<128 THEN 159
158 POKE CLR,0: GOSUB WAIT
159 GOSUB COMPARE
160 IF Y>A(J) THEN 202
163 A(J+1)=A(J)
166 GOSUB INTERCHANGE
168 POKE 50,63
175 X=J: GOSUB DISPLAY: GOSUB DELAY
178 X=J+1: GOSUB DISPLAY: GOSUB
    DELAY
180 POKE 50,255: GOSUB DISPLAY:
     GOSUB DELAY
185 X=J: GOSUB ERASE
200 NEXT J
202 A(J+1)=Y
203 POKE 50,63:X=J+1: GOSUB DISPLAY
204 GOSUB INTERCHANGE
205 POKE 50,255: GOSUB DISPLAY
206 NEXT I
208 VTAB 24: TAB 15: PRINT "FINISHE
    ";
210 IF PEEK (KBD)<128 THEN 210
220 POKE CLR,0: CALL -936: GOTO
    20
500 TEXT : CALL -936
510 VTAB 1: FOR I=0 TO 9: TAB 7
    +3*I: PRINT I;: NEXT I
515 VTAB 2: TAB 7: FOR I=0 TO 9
    : PRINT "---";: NEXT I
520 FOR J=0 TO 9: VTAB 3+2*J: TAB
    4: PRINT J;"! ";: NEXT J
525 VTAB 23: TAB 13: PRINT "INSERTIO
    N SORT"
530 RETURN
600 COL=X MOD 10
610 ROW=X/10
620 VTAB 2*ROW+3: TAB 7+3*COL
630 IF A(X)<10 THEN PRINT " ";
635 PRINT A(X);
640 RETURN
650 COL=X MOD 10:ROW=X/10
655 VTAB 2*ROW+3: TAB 7+3*COL
660 PRINT "  ";
670 RETURN
800 IF KEY<> ASC("Q") THEN 810
805 TEXT : CALL -936: END
810 KEY= PEEK (KBD): IF KEY<128
     THEN 810
820 POKE CLR,0: RETURN
900 REM  *** TO REMOVE SOUND FOR COM
    PARISONS - INSERT 901 RETURN ***

902 POKE PITCH,10: POKE TIME,5:
     CALL MUSIC
905 GOSUB DELAY
910 RETURN
950 REM  *** TO REMOVE SOUND FOR INT
    ERCHANGES - INSERT 951 RETURN **
    *
952 POKE PITCH,49: POKE TIME,3:
     CALL MUSIC
955 GOSUB DELAY
960 RETURN
975 FOR DE=1 TO PDL (1): NEXT DE
980 RETURN
1000 VTAB 10: TAB 5: PRINT "I WILL SO
     RT UP TO 100 POSITIVE"
1001 TAB 5: PRINT "INTEGERS INTO ASCE
     NDING"
1002 TAB 5: PRINT "ORDER USING THE IN
     SERTION SORT."
1008 VTAB 15: TAB 10: INPUT "VALUE OF
      N PLEASE",NUM:N=NUM-1
1010 IF N>=0 THEN 1013
1012 TEXT : CALL -936: END
1013 IF NUM<=100 THEN RETURN
1015 TAB 10
1020 PRINT "TOO BIG!!!!!": GOTO
      1000
```

*Listing 3*
**SELECTION SORT**

```
>PR#0
>LIST
    0 I=J=Y=N
    5 DIM A(99)
    6 KBD=-16384:CLR=-16368:TITLE=
      500:INTRO=1000
    7 DISPLAY=600:WAIT=800:CMP=900
      :INT=950
    8 MUSIC=-10473:TIME=766:TIMBRE=
      765:PITCH=767
    9 DELAY=975:ERASE=650
   10 TEXT : CALL -936
   20 GOSUB INTRO
   50 GOSUB TITLE
  100 FOR I=0 TO N
  105 A(I)= RND (100):X=I: GOSUB
      DISPLAY
  110 NEXT I
  150 FOR I=0 TO N-1
  151 MAX=0
  152 VTAB 23: TAB 32: PRINT "I="
      ;: IF I<10 THEN PRINT " ";:
       PRINT I
  155 FOR J=1 TO N-I
  156 KEY= PEEK (KBD): IF KEY<128
       THEN 158
  157 POKE CLR,0: GOSUB WAIT
  158 GOSUB DELAY
  159 GOSUB CMP
  160 IF A(J)<=A(MAX) THEN 200
  163 MAX=J
  165 VTAB 24: TAB 32: PRINT "M="
      ;: IF MAX<10 THEN PRINT " "
      ;: PRINT MAX;
  168 POKE 50,63
  175 X=J: GOSUB DISPLAY
  178 POKE 50,255
  185 X=J: GOSUB DISPLAY
  200 NEXT J
  202 TEMP=A(MAX): GOSUB INT
  203 A(MAX)=A(N-I):X=MAX: POKE 50
      ,63: GOSUB DISPLAY: GOSUB INT:
       POKE 50,255: GOSUB DISPLAY
  204 A(N-I)=TEMP:X=N-I: POKE 50,
      63: GOSUB DISPLAY: GOSUB INT:
       POKE 50,255: GOSUB DISPLAY
  212 NEXT I
  215 VTAB 24: TAB 15: PRINT "FINISHED
              ";
  218 IF PEEK (KBD)<128 THEN 218
  220 POKE CLR,0: CALL -936: GOTO
      20
  500 TEXT : CALL -936
  510 VTAB 1: FOR I=0 TO 9: TAB 7
      +3*I: PRINT I;: NEXT I
  515 VTAB 2: TAB 7: FOR I=0 TO 9
      : PRINT "---";: NEXT I
  520 FOR J=0 TO 9: VTAB 3+2*J: TAB
      4: PRINT J;"! ";: NEXT J
  525 VTAB 23: TAB 13: PRINT
                    "SELECTIO

       N SORT"
  530 RETURN
  600 COL=X MOD 10
  610 ROW=X/10
  620 VTAB 2*ROW+3: TAB 7+3*COL
  630 IF A(X)<10 THEN PRINT " ";
  635 PRINT A(X);
  640 RETURN
  800 IF KEY# ASC("Q") THEN 810
  805 TEXT : CALL -936: END
  810 IF PEEK (KBD)<128 THEN 810
  815 POKE CLR,0
  849 RETURN
  900 REM  *** TO REMOVE SOUND FOR COM
      PARISONS - INSERT 901 RETURN ***

  902 POKE PITCH,10: POKE TIME,5:
       CALL MUSIC
  905 GOSUB DELAY
  910 RETURN
  950 REM  *** TO REMOVE SOUND FOR INT
      ERCHANGES - INSERT 951 RETURN **
      *
  952 POKE PITCH,49: POKE TIME,3:
       CALL MUSIC
  955 GOSUB DELAY
  960 RETURN
  975 FOR DE=1 TO PDL (1): NEXT DE
  999 RETURN
 1000 VTAB 10: TAB 5: PRINT "I WILL SO
      RT UP TO 100 POSITIVE"
 1001 TAB 5: PRINT "INTEGERS INTO ASCE
      NDING"
 1002 TAB 5: PRINT "ORDER USING THE SE
      LECTION SORT."
 1008 VTAB 15: TAB 10: INPUT "VALUE OF
       N PLEASE",N
 1010 IF N>0 THEN 1013
 1011 TEXT : CALL -936: END
 1013 IF N<=99 THEN RETURN
 1015 TAB 10
 1020 PRINT "TOO BIG!!!!!": GOTO
      1000
```

## *Listing 4*
## SHELL SORT

```
>PR#0
>LIST
 100  DIM A(99),INCS(5)
 105  MUSIC=-10473:PITCH=767:TIME=
      766:TIMBRE=765: POKE TIMBRE,
      32
 110  KBD=-16384:CLR=-16368:TITLE=
      400:INTRO=1000
 120  DISPLAY=500:WAIT=800:CMP=900
      :INT=950
 125  DELAY=975:ERASE=550
 130  TEXT : CALL -936
 140  GOSUB INTRO
 150  GOSUB TITLE
 160  FOR I=0 TO N
 170  A(I)= RND (100):X=I: GOSUB
      DISPLAY
 180  NEXT I
 190  INCS(1)=10:INCS(2)=6:INCS(3
      )=4:INCS(4)=2:INCS(5)=1
 200  FOR I=1 TO 5
 210  SPAN=INCS(I)
 211  IF SPAN>N THEN 370
 215  VTAB 24: TAB 12: PRINT "SPAN="
      ;
 216  IF SPAN<10 THEN PRINT " ";:
       PRINT SPAN;
 220  FOR J=SPAN TO N
 230  Y=A(J): GOSUB INT
 233  VTAB 23: TAB 28: PRINT "J= "
      ;: IF J<10 THEN PRINT " ";:
       PRINT J
 235  TAB 26: PRINT "A(J)=";: IF
      A(J)<10 THEN PRINT " ";
 236  POKE 50,63: PRINT A(J);: POKE
      50,255
 240  FOR K=J-SPAN TO 0 STEP -SPAN
 245  GOSUB CMP
 250  IF Y>A(K) THEN 320
 260  POKE 50,63
 265  GOSUB INT
 270  A(K+SPAN)=A(K)
 280  X=K+SPAN: GOSUB DISPLAY
 285  KEY= PEEK (KBD): IF KEY<128
       THEN 290
 287  POKE CLR,0: GOSUB WAIT
 290  GOSUB DELAY
 300  POKE 50,255: GOSUB DISPLAY
 305  X=K: GOSUB ERASE
 310  NEXT K
 320  POKE 50,63
 325  GOSUB INT
 330  A(K+SPAN)=Y:X=K+SPAN: GOSUB
      DISPLAY
 340  GOSUB DELAY
 350  POKE 50,255: GOSUB DISPLAY
 360  NEXT J
 370  NEXT I
 380  VTAB 24: TAB 12: PRINT "FINISHE
      "
 390  IF PEEK (KBD)<128 THEN 390
 395  POKE CLR,0: CALL -936: GOTO
      140
 400  TEXT : CALL -936
 420  VTAB 1: FOR I=0 TO 9: TAB 7
      +3*I: PRINT I;: NEXT I
 430  VTAB 2: TAB 6: FOR I=0 TO 9
      : PRINT "---";: NEXT I
 440  FOR J=0 TO 9: VTAB 3+2*J: TAB
      4: PRINT J;"! ";: NEXT J
 450  VTAB 23: TAB 10: PRINT " SHELL S
      ORT"
 460  RETURN
 500  COL=X MOD 10
 510  ROW=X/10
 520  VTAB 2*ROW+3: TAB 7+3*COL
 530  IF A(X)<10 THEN PRINT " ";
 540  PRINT A(X);
 549  RETURN
 550  COL=X MOD 10:ROW=X/10
 555  VTAB 2*ROW+3: TAB 7+3*COL
 560  PRINT "  ";
 599  RETURN
 800  IF KEY<> ASC("Q") THEN 810
 805  TEXT : CALL -936: END
 810  KEY= PEEK (KBD): IF KEY<128
       THEN 810
 820  POKE CLR,0: RETURN
 900  REM  *** TO REMOVE SOUND FOR COM
      PARISONS - INSERT 901 RETURN ***

 902  POKE PITCH,10: POKE TIME,3:
       CALL MUSIC
 905  GOSUB DELAY
 949  RETURN
 950  REM  *** TO REMOVE SOUND FOR INT
      ERCHANGES - INSERT 951 RETURN **
      *
 952  POKE PITCH,49: POKE TIME,3:
       CALL MUSIC
 955  GOSUB DELAY
 960  RETURN
 975  FOR DE=1 TO PDL (1): NEXT DE
 999  RETURN
1000  VTAB 10: TAB 5: PRINT "I WILL SO
      RT UP TO 100 POSITIVE"
1010  TAB 5: PRINT "INTEGERS INTO ASCE
      NDING"
1020  TAB 5: PRINT "ORDER USING THE SH
      ELL SORT"
1030  VTAB 15: TAB 10: INPUT "VALUE OF
       N PLEASE",N
1040  IF N>0 THEN 1060: CALL -936
      : END
1060  IF N<=99 THEN RETURN
1070  TAB 10
1080  PRINT "TOO MANY!!!!!": GOTO
      1000
2000  POKE CLR,0
2010  KEY= PEEK (KBD): IF KEY<128
       THEN 2010
2020  POKE CLR,0: RETURN
```

**Listing 5**
**QUICKSORT**

```
>LIST
  5 DIM A( 200 ),STACK( 24 )
  6 KBD=-16384:CLR=-16368:TITLE=
    5000:INTRO=10000
  7 DISPLAY=6000:CMP=6500:DELAY=
    6600
  8 MUSIC=-10473:TIME=766:TIMBRE=
    765:PITCH=767
 10 TEXT : CALL -936
 20 GOSUB INTRO
 50 GOSUB TITLE
100 FOR I=0 TO N
105 A( I )= RND (100 ):X=I: GOSUB
    DISPLAY
110 NEXT I
115 A( N+1 )=32767
120 P=0:Q=N
125 TOP=0:MAXTP=0
130 IF P>=Q THEN 170
135 K=Q+1
137 VTAB 23: TAB 34: PRINT "P= "
    ;: IF P<100 THEN PRINT " ";
    : IF P<10 THEN PRINT " ";: PRINT
    P
138 TAB 34: PRINT "Q= ";: IF K<
    100 THEN PRINT " ";: IF K<10
     THEN PRINT " ";: PRINT K;
139 GOSUB 1145
140 IF J-P<Q-J THEN 150
143 GOSUB 400
144 GOTO 160
150 GOSUB 500
160 TOP=TOP+2
161 IF TOP>MAXTP THEN MAXTP=TOP
162 VTAB 24: TAB 23: PRINT (TOP/
    2);
163 IF PEEK (KBD)>=128 THEN GOSUB
    8000
165 GOTO 130
170 IF TOP=0 THEN 208
175 Q=STACK( TOP ):P=STACK( TOP-1 )
    :TOP=TOP-2
176 GOSUB 7500
177 VTAB 24: TAB 23: PRINT (TOP/
    2);
179 IF PEEK (KBD)>=128 THEN GOSUB
    8000
180 GOTO 130
208 VTAB 24: TAB 4: PRINT "FINISHED"
    ;
209 TAB 15: PRINT "MAXTOP= ";(MAXTP/
    2);
210 IF PEEK (KBD)<120 THEN 210
220 POKE CLR,0: CALL -936: GOTO
    20
400 STACK( TOP+1 )=P
405 STACK( TOP+2 )=J-1
410 P=J+1
415 GOSUB 7000
499 RETURN
500 STACK( TOP+1 )=J+1
505 STACK( TOP+2 )=Q
510 Q=J-1
515 GOSUB 7000
599 RETURN
1145 V=A( P ):I=P:J=K
1160 J=J-1: IF A( J )<=V THEN 1170

1162 GOSUB DELAY
1165 GOSUB CMP: GOTO 1160
1170 I=I+1: IF A( I )>=V THEN 1180

1172 GOSUB DELAY
1175 GOSUB CMP: GOTO 1170
1180 IF J<=I THEN 1200
1185 TEMP=A( I )
1186 A( I )=A( J ):X=I: GOSUB DISPLAY
1188 A( J )=TEMP:X=J: GOSUB DISPLAY
1195 IF PEEK (KBD)<128 THEN 1160

1196 GOSUB 8000
1199 GOTO 1160
1200 A( P )=A( J ):X=P: GOSUB DISPLAY
1202 A( J )=V:X=J: GOSUB DISPLAY
1999 RETURN
5000 TEXT : CALL -936
5010 VTAB 1: FOR I=0 TO 9: TAB 7
     +3*I: PRINT I;: NEXT I
5020 VTAB 2: TAB 7: FOR I=0 TO 9
     : PRINT "---";: NEXT I
5030 FOR J=0 TO 19: VTAB 3+J: TAB
     3
5035 IF J<10 THEN PRINT " ";: PRINT
     J;"! ";: NEXT J
5040 VTAB 23: TAB 3: PRINT "QUICKSORT
        PARTITION=======>"
5045 VTAB 24: TAB 15: PRINT "PENDING:
     0";
5050 VTAB 5: TAB 39: PRINT "S": TAB
     39: PRINT "T": TAB 39: PRINT
     "A": TAB 39: PRINT "C": TAB
     39: PRINT "K"
5060 FOR R=10 TO 22: TAB 39: PRINT
     ".": NEXT R
5099 RETURN
6000 COL=X MOD 10
6010 ROW=X/10
6020 POKE 50,63
6030 VTAB ROW+3: TAB 7+3*COL
6040 IF A( X )<10 THEN PRINT " ";
6050 PRINT A( X );
6060 POKE 50,255
6070 VTAB ROW+3: TAB 7+3*COL
6080 IF A( X )<10 THEN PRINT " ";
6090 PRINT A( X );
6100 REM  *** TO REMOVE SOUND FOR INT
     ERCHANGES - INSERT 6101 RETURN *
     **
6110 POKE PITCH,49: POKE TIME,3:
      CALL MUSIC
```

```
6199 RETURN
6500 REM  *** TO REMOVE SOUND FOR COM
     PARISONS - INSERT 6501 RETURN **
     *
6510 POKE PITCH,10: POKE TIME,5:
      CALL MUSIC
6599 RETURN
6600 FOR DE=0 TO PDL (1): NEXT DE
6699 RETURN
7000 VTAB 21-TOP: TAB 37
7005 TOS=STACK(TOP+1):NOS=STACK(
     TOP+2)
7010 IF NOS<100 THEN PRINT " ";:
      IF NOS<10 THEN PRINT " ";:
      PRINT NOS
7015 TAB 37: IF TOS<100 THEN PRINT
     " ";: IF TOS<10 THEN PRINT
     " ";: PRINT TOS;
7499 RETURN
7500 VTAB 21-TOP: TAB 37: PRINT
     "   ": TAB 37: PRINT "   ";
7999 RETURN
8000 POKE CLR,0
8005 IF PEEK (KBD)<128 THEN 8005

8010 POKE CLR,0
8099 RETURN
10000 VTAB 10: TAB 5: PRINT "I WILL SO
      RT UP TO 100 POSITIVE"
10010 TAB 5: PRINT "INTEGERS INTO ASCE
      NDING"
10020 TAB 5: PRINT "ORDER USING HOARE'
      S QUICKSORT."
10030 VTAB 15: TAB 10: INPUT "VALUE OF
       N PLEASE",N
10040 IF N>0 THEN 10060
10050 TEXT : CALL -936: END
10060 IF N<=199 THEN RETURN
10070 TAB 10
10080 PRINT "TOO BIG!!!!!": GOTO
      10000
```

# “Hello, World”

**A very Inexpensive anelog interfece is presented that cen be used with eny microcomputer. Some PET oriented progrems ere provided, including a STAR ACE geme, to show how the device may be utilized.**

**John Sherburne**

When I bought my PET,one of the things I eventually wanted to do was to interface the computer to the outside world. Over the two years since then I have seen interface devices of one kind or another,but all of them have been fairly expensive, and most are designed for a single application. I have finaly found one interface, however, which is cheap, simple enough for even the laziest Sunday solderer to build, and is useful for a variety of real world applications. By plugging in a joystick or two, arcade-type games can be created. If the interface is used to dense switch settings, educational programs or game show recreations can be easily made. Adding a potetiometer or thermistor as a sensor permits measurement of temperature, wind direction or other external conditions. All in all, it is the best way I have found for the PET owner with a tight budget to branch out into new areas.

The interface uses a single integrated circuit — an NE555 timer. The principle of operation is to hook up the timer as in Figure 1 so that it emits a pulse when triggered by the PET. The duration of the pulse depends upon the magnitude of the resistance, R1, in the circuit. By timing the pulse duration with the PET internal clock, the resistance can be measured. Thus, any device which translates an external quality into a resistance can be used as a sensor. Using the circuit requires three elements: a 5 volt DC power supply, the 555-based timer and a sensor.If you don't already have a power supply there is no need to buy an expensive one just for this application. I found that a small kit such as the Jameco JE 200 is adequate, inexpensive ($14.95) and can be put together in less than an hour.As for sensors, the cost and availability depend on what you want to do. A simple measure of displacement can be made with a potentiometer costing less than a dollar.Precision probes for temperature, on the other hand, may be expensive and hard to find. The third element, the NE555, costs about 60¢ and a four timer Interface with board, wire, connectors and the like can be constructed for about $10.

Interface to the PET Is made through pins PA0 - PA7 of the parallel user port shown In Figure 2. These eight pins can be programmed for either input or output by changing the contents of memory location 59459 (E843). If bit n of that location is a zero, PAn will be an input pin. If bit n is a one, PAn will be an output pin. For example, POKE 59459,15 will make pins PA0 — PA3 output and pins PA4 — PA7 input. Once programmed, the pins are read or driven via location 59471 (E84F). In this way the user port can be programmed so that one pin is used as output to trigger a 555 and another pin is used as input to sense the duration of the timer pulse. Since there are eight pins, four 555s can be connected without resorting to encode/decode arrangements.

Figure 3 is a schematic of a four 555 interface. The interface is sufficient to handle two joysticks — each of which has two potentiometers or four individual sensors. Two NE556s could also be used since the 556 is a dual 555. The pin by pin connection for each of the 555s is as follows:

1 Connect to ground.

2 Trigger.Connect to output pin of users port. This pin is normally high (+5V). When brought momentarily to ground, it starts the 555 output pulse.

3 Output. Connect to users port input pin. This pin is normally low (ground). During the output pulse it is high.

4 Connect to +5V.

5 Connect to ground through bypass capacitor C2

6 Connect to +5V through sensor R1 and connect to ground through timing capacitor C1.

7 Connect to pin 6.

8 Connect to +5V.

Each of the four 555s in Figure 3

is connected the same way. The four trigger pins (pin 2) are connected to PA0 — PA3 and the four output pins (pin 3) are connected to PA4 — PA7. The PET ground is connected through R2 to the IC ground (pin 1).

The output pulse duration of the 555 is dependent both on R1 and C1. As C1 is increased in capacitance, the pulse is longer. A .01 yf capacitor works well for moderate sensor resistances (50K to 1 meg ohm). For lower resistances, a higher capacitance is needed. Capacitors must be high quality mylar for stability. The duration of the output pulse also increases as R1 increases. If there is no resistance at R1, that is, pin 7 is shorted to +5V, the pulse duration will be essentially zero. An open circuit between pins 5 and 7 will cause an almost unending pulse.

To measure the duration of the pulse, one of the timers associated with the parallel user port is accessed. The timer is two bytes long and decrements with every cycle of the PET clock (every microsecond). The least significant byte of the timer is at location 59464 (E848). It starts at 255, counts down to zero and recycles. The most significant byte is 59465. It starts at 255 and counts down each time 59464 reaches zero. The speed of the timer requires that machine language rather than BASIC be used to access it. Program 1 is a simple assembly language program which drives one pin of the user port low then high, starts the timer and waits for the end of the output pulse of the 555. The pulse length is then stored in locations 42 and 43 (2A and 2B). The pins to be used for output and input are determined by memory locations 40 and 41 (28 and 29), respectively. For example, if bit 6 of location 41 is a one, then it takes 16 clock cycles to start the output pulse and check the input pin, 16 microseconds is the minimum pulse width that can be measured in increments of 7 cycles beginning at 16 (16,23,30...).

Once the interface has been constructed, Program 1 can be used to test its operation. First connect pin 6 of each 555 to +5V, then load Program 1 and key in the following:

```
10 POKE 59459,15
20 FOR I=0 TO 3
30 POKE 40, 16*2 I:POKE 41,2 I;SYS(977)
40 A=255-PEEK (42) +256*(255-PEEK(43))
50 PRINT A: NEXT
```

The result should be that A is about equal to the minimum 16 in each case. The program assumes that four 555s are present with pin 2 of each connected to one of the first four pins of the user port. Pin 3 of each 555 is connected to one of the last four pins of the user port. That is, if pin 2 of a 555 is connected to PAn, then pin 3 is connected to PAn+4. If there is a mistake in wireing or software the result will probably be a list cursor type crash.

The easiest sensor to connect in the circuit is a simple switch. If a 50K resistor is connected across the poles of the switch, the switch will present no resistance in one position and a resistance of 50K resistor is connected across the poles of the switch, the switch will present no resistance in one position and a resistance of 50L in the other position. Connecting four such switches in series with a different resistance across each one enables the 555 to determine which of the four switches has been thrown. If normally closed pushbuttons are used with resistances of 50K, 150K, 300K and 600K as buttons are pushed, a resistance of 50K when button #1 is pushed, 150K for #2, 200K for *#1 and* #2, and so forth. This arrangement can be used as the basis for quiz or educational games where the players give their answers by pushing one of the buttons. Since only one 555 is required for each set of switches, up to four players can play at the same time.

Another useful switch arrangement is to connect a normally open pushbutton in place of R1 for each 555. If a 555 is triggered it will emit an output pulse which will continue until its pushbutton is pressed. A test of reflex speed can be constructed by triggering all four 555s, instruction the player to push one of the buttons and then measuring the time it takes him to respond.

Since the response time will be longer than the timer at 59464 can handle, the "jifty" timer, TI, should be used. Program 2 is an example of how the timer can be used. The recheck procedure in lines 220 and 230 is needed to correct for poor pushbutton action. The value Z in line 165 should be set to yield Y*0 when there is no time delay between asking for a response and pushing the button. The same principle used in the reflex test can be used along with CB2 sound to simulate the electronic games which require the duplication of a series of sounds.

One of the more useful applications of the 555 Interface is the joystick. One 555 is used to sense the position of each of the two potentiometers in the joystick. There are two ways that the joystick position can be translated into cursor movement. One is to move the cursor relative to some fixed position such as the center of the screen. In this mode a given joystick position always moves the cursor to the same spot on the screen. The technique is useful in obtaining input for games like Checkers or Othello. The other mode is to use the joystick position to indicate movement relative to the current postion of the cursor.This technique is useful in manuevering through a maze or in other real-time games. In this mode moving the joystick in a given direction moves the cursor in that direction. As long as the joystick is held in that positsion the cursor will continue to move. Returning the joystick to the center stops the cursor. The following sequence illustrates this technique:

"DOODLE"

```
10 RT=20:UP=12
20 POKE 59459,15
30 REM   CALIBRATE JOYSTICK IN CENTER
40 PRINT "[clear]PLACE JOYSTICK IN CENTER.  PRE
SS ANY KEY WHEN READY."
50 GET A$: IF A$="" GOTO 50
60 POKE 40,16:POKE 41,1:SYS(977)
70 A=255-PEEK(42)+256*(255-PEEK(43))
80 POKE 40,32:POKE 41,2:SYS(977)
90 B=255-PEEK(42)+256*(255-PEEK(43))
100 AL=.6*A:AH=1.2*A
110 BL=.6*B:BH=1.2*B
```

Of course, this routine must be used in conjunction with Program 1. The routine can easily be expanded to move the cursor more than one location at larger joystick displacements. With some checks to keep the print position on the screen added, the program can be used to draw pictures or "doodle".

μ

```
1000 REM  SENSE JOYSTICK POSITION
1010 POKE 40,16:POKE 41,1:SYS(977)
1020 A=255-PEEK(42)+256*(255-PEEK(43))
1030 POKE 40,32:POKE 41,2:SYS(977)
1040 B=255-PEEK(42)+256*(255-PEEK(43))
1050 REM CALCULATE NEW POSITION
1060 R=-1:IF A>AL THEN R=0:IF A>AH THEN
R=1
1070 U=-1:IF B>BL THEN U=0:IF B>BH:THEN
U=1
1080 RT=RT+R:UP=UP+U:PRINT "[home]";
1090 FOR I=1 TO UP:PRINT:NEXT
1100 PRINTTAB(RT) "X":GO TO 1000
```

*John Sherburne is an operations research specialist with the Department of Defense. He has a number of years experience in mathematical computer programming. Microcomputing is his hobby.*

**PROGRAM 1** **Assembly Language**

```
03D1  A5  28             LDA  IPUT   :Load input mask
03D3  A6  29             LDX  OPUT   :Load output mask
03D5  8E  4F  E8         STX  PORT   :Set trigger high
03D8  A0  00             LDY  # 00
03DA  84  2A             STY  ANSR   :Clear result
03DC  84  2B             STY  ANSR+1
03DE  8C  48  E8         STY  TIML   :Clear timer
03E1  8C  49  E8         STY  TIMM   ;Clear & start timer
03E4  8C  4F  E8         STY  PORT   :Bring trigger low
03E7  8E  4F  E8         STX  PORT   :Return to high
03EA  2C  4F  E8    WAIT BIT  PORT   :Wait for end of pulse
03ED  D0  F8             BNE  WAIT
03EF  AE  48  E8         LDX  TIML   :Store result
03F2  AC  49  E8         LDY  TIMM
03F5  86  2A             STX  ANSR
03F7  84  2B             STY  ANSR+1
03F9  60                 RTS
```

**BASIC Program to Load Assembly Language**

```
10 DATA 165,40,166,41,142,79,232,160,0,
132,42,132,43,140,72,232,140,73,232,140

20 DATA 79,232,142,79,232,44,79,232,208
,251,174,72,232,172,73,232,134,42,132

30 DATA 43,96

40 FOR I=977 TO 1017

50 READ A:POKE I,A:NEXT
```

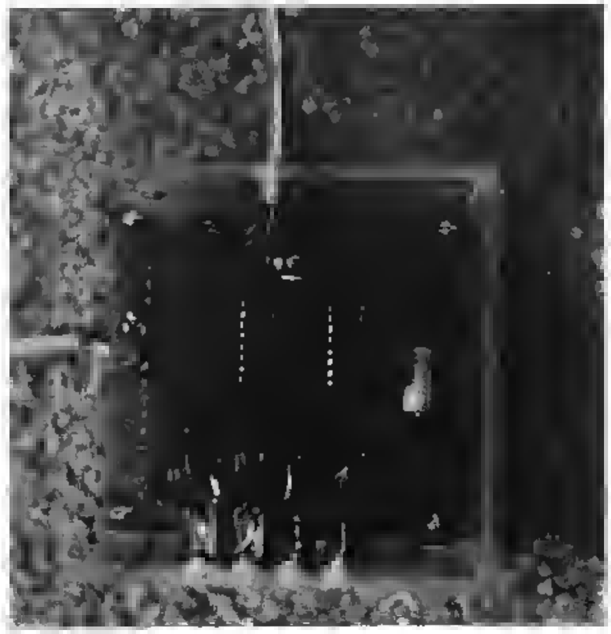
**PROGRAM 2**

```
10 POKE 59459,15:Z=9
20 N(0)=239:N(1)=223:N(2)=191:N(3)=127
25 L$(0)="A":L$(1)="B":L$(2)="C":L$(3)="D"
30 PRINT "[clear]THIS IS A TEST OF YOUR REA
CTION TIME"
31 PRINT "[down]WHEN YOU SEE A LETTER ON THE
 SCREEN"
32 PRINT "[down]PRESS THE BUTTON WITH THE SA
ME LETTER"
33 PRINT "[2 down]PRESS ANY KEY WHEN YOU ARE
READY"
```

**Program 2 cont.**

```
40 GET A$:IF A$="" GOTO 40
60 I=999+INT(500*RND(1))
70 FOR K=0 TO I:NEXT
120 POKE 59471,15
122 I=INT(4*RND(1))
130 TI$="000000":E=0
140 POKE 59471,0
145 PRINT "[down]";L$(I)
150 POKE 59471,15
160 WAIT 59471,255,255
170 R=PEEK(59471)
180 IF R<>N(I) GOTO 220
190 Y=INT(Y*100/60)/100
200 PRINT "YOU TOOK";Y;"SECONDS":END
220 IF E=0 THEN E=1:GOTO 170
230 IF E=1 THEN E=2:POKE 59471,0:GOTO 1
50
300 PRINT "[clear] WRONG BUTTON!":END
```

**Notes:**
Line 140 and line 150 start timerpulse.
Line 160 waits until one of the pins PA4 - PA7 goes low.Line 180 checks to see if proper button was pushed. Lines 220 and 230 recheck for errors caused by poor pushbutton action.



*View of assembler four 555 interface device.*

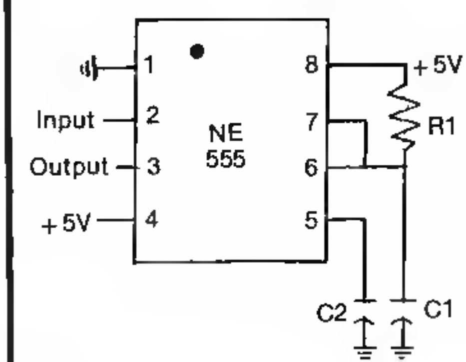
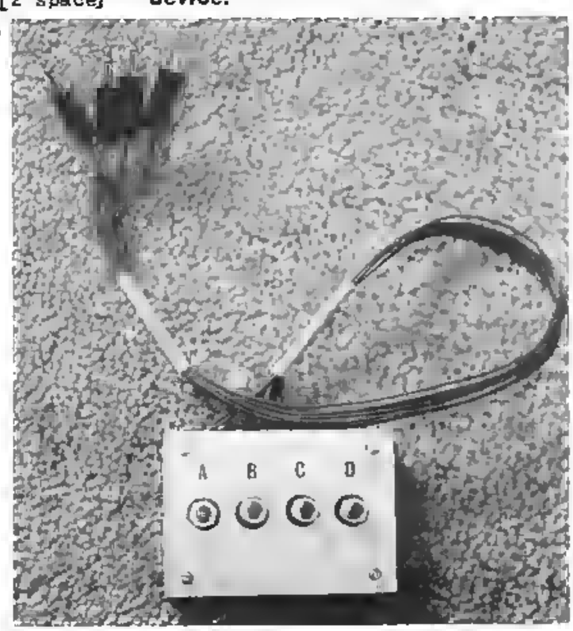



*Figure 1:* **Pinout diagram for the NE555**

## STAR ACE

```
10 DIM DN$(24),FG$(3):POKE 59459,15
20 DATA "","[down]","[2 down]","[3 down]","[4 down]","[5 down]",
"[6 down]","[7 down]","[8 down]"
30 DATA "[9 down]","[10 down]","[11 down]"
,"[12 down]","[13 down]"
40 DATA "[14 down]","[15 down]"
,"[16 down]"
50 DATA "[17 down]","[18 down]"
,"[19 down]"
60 DATA "[20 down]","[21 down]"
,"[22 down]"
70 DATA "[23 down]","[24 down]"
80 DATA "/‾‾‾\[down][back][space][down][back][space][down][back][space]
[down][back]/[back][back]_[back][back]_[back][back]_[back][back]\
[up][back][space][up][back][space]┌┼┐[down][back]+[down][back]┘[back]
[back]└[up][back]+○"
82 DATA "[space]▄─[down][2 back][rvs]▀[down][back]▀[off]─[3 back]▄[up]
[forward]▄▄▄─"
90 FOR I=0 TO 24:READ DN$(I):NEXT
102 READ ST$:READ TG$
110 DATA "[rvs]◥[off]*****[down][4 back]****─","****─[down][5 back]◥*
[rvs]▄[off]","┴*[down][back]\**[down][4 back]◥*[2 space]*[space]|
[3 back]▄"
112 DATA "├[space]*[down][2 back]‾*[down][3 back]◤*[2 space]*/[down]
[3 back]▚","┬[space]**[down][3 back]/*[down][3 back][rvs]◥[off][2 space]
*─[down][3 back]▌"
114 DATA "┤[space]**[down][3 back]_[space]*[down][4 back][rvs]◤
**[space]\[down][3 back][rvs]▚[off]"
                                                            [off]
120 FOR I=0 TO 5:READ E$(I):NEXT
180 PRINT "[clear][3 space]YOUR SHIP IS UNDER ATTACK BY ENEMY"
181 PRINT "FIGHTERS.  THE ENEMY FIGHTERS WILL BE"
182 PRINT "IN RANGE FOR ONLY TWO MINUTES!  YOU"
183 PRINT "MUST DESTROY AS MANY AS POSSIBLE WHILE"
184 PRINT "CONSERVING LASER POWER FOR FUTURE USE"
185 PRINT "[down][3 space]USE THE JOYSTICK TO AIM YOUR LASER."
186 PRINT "[down][3 space]PRESS 'F' TO FIRE."
187 PRINT "[down][3 space]PLACE JOYSTICK IN CENTER POSITION"
188 PRINT "AND PRESS ANY KEY TO BEGIN. . GOOD LUCK!"
210 GET A$:IF A$ = "" GOTO 210
220 POKE 40,16:POKE 41,1:SYS(977)
230 A=255-PEEK(42)+256*(255-PEEK(43))
240 POKE 40,32:POKE 41,2:SYS(977)
250 B=255-PEEK(42)+256*(255-PEEK(43))
260 A1=.3*A:B1=.3*B
261 A2=.7*A:B2=.7*B
262 A3=1.3*A:B3=1.3*B
263 A4=1.7*A:B4=1.7*B
280 HI=0:SH=0:LM=TI
290 DY=12:RX=0:HO=20:VE=12
295 FOR I=1 to 999:NEXT:PRINT "clear "
300 Y=DY+RND(1)-.5:X=RX+2*RND(1)
310 IF Y<2 THEN Y=2
312 IF Y>21THEN  Y=21
314 IF X>35THEN  PRINT "[clear]":GOTO 290
```

*View of assembled reflex testing device.*



```
400 POKE 40,16:POKE 41,1:SYS(977)
410 A=255-POKE(42)+256*(255-POKE(43))
420 POKE 40,32:POKE 41,2:SYS(977)
430 B=255-POKE(42)+256*(255-POKE(43))
440 H=2:IF A>A1 THEN H=1:IF A>A2 THEN H=0:IF A>A3 THEN
H=-1:IF A>A4 THEN H=-2
450 V=2:IF B>B1 THEN V=1:IF B>B2 THEN V=0:IF B>B3 THEN
V=-1:IF B>B4 THEN V=-2
460 H=HO+H:V=VE+V
461 IF V>19 THEN V=19
462 IF H>35 THEN H=35
464 IF H<0 THEN H=0
466 IF V<0 THEN V=0
520 PRINT "[clear]";DN$(V)TAB(H)ST$
530 PRINT "[home]";DN$(Y)TAB(X)TG$
535 IF TI-LM>7200 GOTO 700
540 HO=H:VE=V:DY=Y:RX=X
```

STAR ACE requires use of a joystick and the assembly language interface programs. Brackets, [], are used to show special characters. For example, [3down] means three down cursor characters.

```
550 GET A$:IF A$<>"F" GOTO 300
555 PRINT "[home]";"LASER'S FIRED!":SH=SH+1
556 C=PEEK(32580+40*V+H)
560 IF C<>98 AND C<>254 GOTO 300
565 PRINT "[clear]";DN$(Y)TAB(X)E$(0)
570 PRINT "[clear]";DN$(Y)TAB(X)E$(1)
575 FOR I=1 TO 4
580 FOR J=2 TO 5
590 PRINT "[clear]";DN$(Y+I)TAB(X)E$(J)
595 NEXT J:NEXT I
600 HI=HI+1:PRINT "[clear]HITS ";HI:PRINT "SHOTS FIRED ";SH
610 GOTO 290
700 SC=100*HI-(10*SH)
710 PRINT "YOUR SCORE IS ";SC
720 IF SC>499 THEN PRINT "[3 down] ACE!!!! CONGRATULATIONS.":
END
730 IF SC>249 THEN PRINT "[3 down]GOOD SHOOTING!":END
740 IF SC>0 THEN PRINT "[3 down]YOU NEED MORE PRACTICE":END
750 IF SC<1 THEN PRINT "[3 down]YOU'RE LUCKY TO STILL BE
ALIVE":END
```

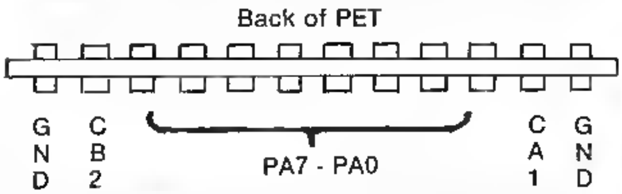



***Figure 2:* Rear view of the PET Parallel User Port. All pins are on the bottom of the edge card. PA0 is to the right.**

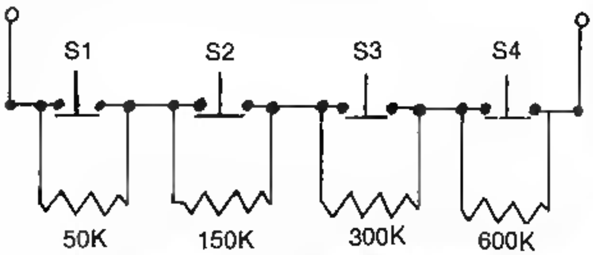



***Figure 4:* Schematic of a response sensing device.**

***Screen display from STAR ACE game.***

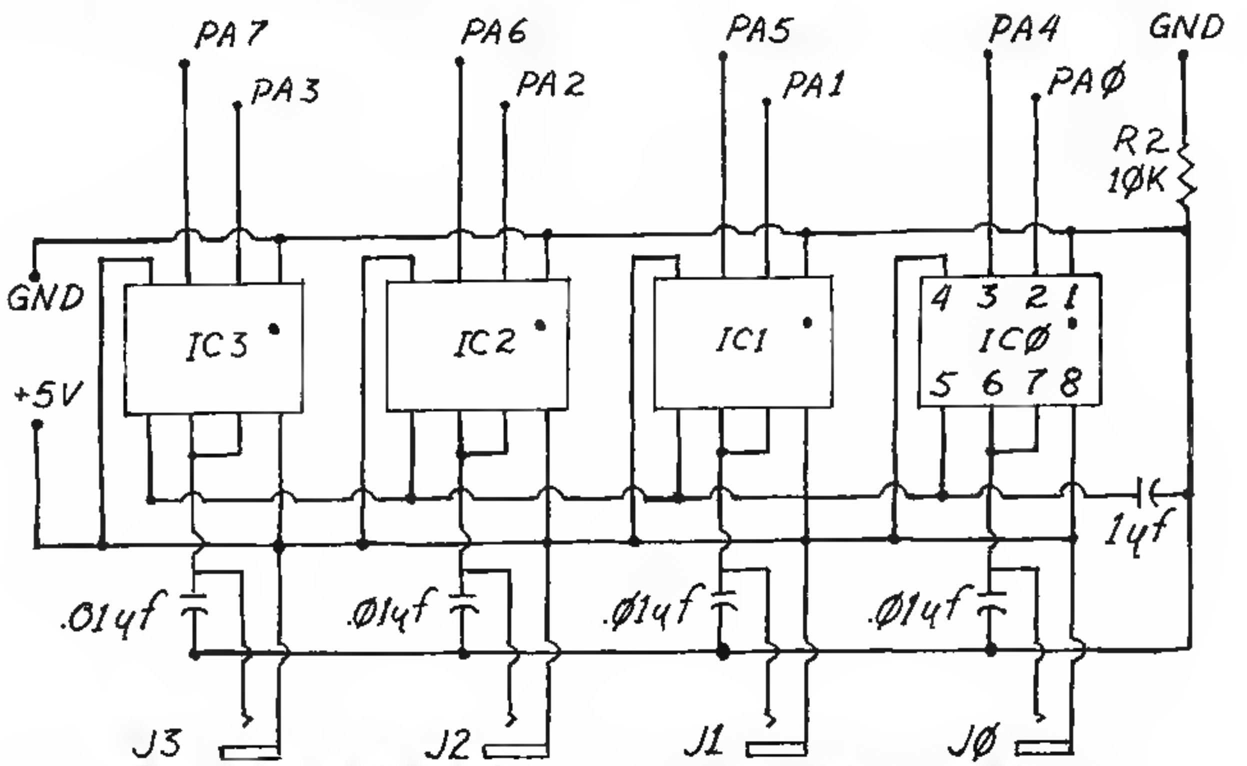
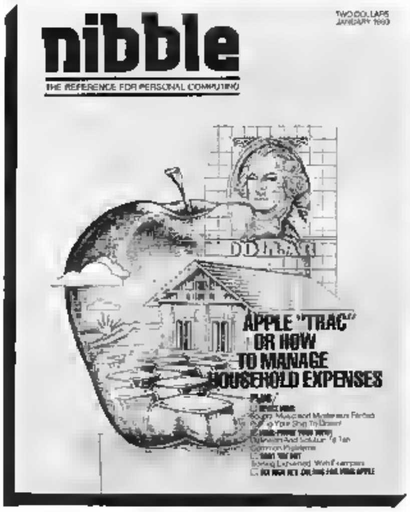



***Figure 3:* Schematic of a four device interface. Connections to the computer are at the top. Jacks J0 to J3 are phone jacks for connecting sensors. All capacitors are Mylar.**

# INTRODUCING . . . NIBBLE
# THE REFERENCE FOR APPLE COMPUTING




**NIBBLE IS:**
*A SOFTWARE GUIDE* for high quality Applications Programs for your Home and Business.

**NIBBLE IS:**
*A REFERENCE GUIDE* to new Programming Methods.

**NIBBLE IS:**
*A BUYERS GUIDE* for making purchase decisions on new products.

**NIBBLE IS:**
*A CONSTRUCTION PROJECT COOKBOOK* for adding function and value to the system you already own.

**NIBBLE IS:**
*A COMMUNICATIONS CLEARING HOUSE* for users, vendors, and associations.

Each issue of NIBBLE features at least one significant new application program of commercial quality. The programs in NIBBLE are surrounded with articles which show how to USE the programming methods in your OWN programs.

Examples of upcoming articles:
☐ Modeling and Forecasting Your Business ☐ Build a Two-Tape Controller for $12 ☐ Arcade Shooting Gallery — Save Your Quarters! ☐ Data Base Management System I, II, III

And many many more! NIBBLE will literally "Nibble Away" at the mysteries of your system to help you USE IT MORE. In 1980, the principal featured system is the Apple II.

**Try a NIBBLE**

**nibble**
BOX 325 Lincoln, Mass. 01773

**I'll try NIBBLE!**
**Enclosed is my $15 for 8 issues.**
☐ **check** ☐ **money order**

Name ______________________

Address ______________________

City ______________________

State ______________ Zip ________

# Zoom And Squeeze

**A short program for tha Apple II which makas it easiar to adit BASIC programs. ZOOM provides e fast wey to copy ovar e program line; SQUEEZE chengas the screen width to 33 cherectars and eliminates ambadded blanks.**

**Gary B. Little**

ZOOM and SQUEEZE is a short machine-language routine written for the APPLE microcomputer in order to facilitate the editing of BASIC programs. It recognizes two commands: CTRL-Q and CTRL-Z. The CTRL-Q command causes the screen window width to be automatically set to 33 and the CTRL-Z command causes the cursor to quickly copy over all text from its current position to the end of the line.

### The ZOOM Feature

In order to edit a program line on the APPLE it is necessary to more than simply move the cursor directly to the area to be changed, make the changes, and then press RETURN-the required procedure is to position the cursor at the beginning of the line number, copy down to the area to be changed (by using the right-arrow and repeat keys, make the changes, and enter the edited line. If the line is a very long one, the copying-over part of this procedure takes up an enormous amount of time which can be better used for other purposes.

The 'ZOQM' part of the ZOOM and SQUEEZE routine can be used to speed up this copying tremendously. By simply pressing CTRL-Z the cursor can be moved virtually instantaneously from its current position to the right edge of the current line while automatically copying over all the text on the screen in between. For example, to copy over a program line that takes up three lines on the video screen takes only six quick steps after the cursor has been positioned at the beginning of the line number: CTRL-Z, right-arrow, CTRL-Z, right-arrow, CRLT-Z, RETURN. This takes approximately 2 seconds to accomplish. By way of contrast, to copy over the line in the ordinary way by using the right-arrow key in conjunction with the repeat key takes aproximately 13 seconds (see the NOTE below!)

It is clear, then, that this feature could save hours of debugging time for a busy programmer.

### The SQUEEZE Feature

When a line of a BASIC program is listed on the video screen with the window width set at its default value of 40 columns, the output is carefully formatted by the APPLE by embedding blanks on the left and right sided of the listing. That is to say, there is not a continuous 'wrap-around' display of the information that you typed in to create the line. For example, if you enter the line

```
100 PRINT "THIS IN AN EXAMPLE
OF A FORMATTED LISTING"
```

and then LIST it, the APPLE will respond with

```
100 PRINT "THIS IS AN EXAMPLE
OF A F**
****ORMATTED LISTING"
```

where a '*' Indicates an embedded blank. This formatting technique makes it very easy to read a LISTed line, but it can create a minor problem when it becomes necessary to edit the line.

The problem arises when, as In the example, the blanks are embedded between the quotation marks associated with a PRINT statement. If this line is to be edited without retyping it from scratch, the right-arrow key (in conjunction with the repeat key) must be used to copy over substantial portions of the line and by so doing all 6 of the embedded blanks between 'F' and 'QRMATTED' will mysteriously appear in the argument of the PRINT statement UNLESS they are skipped over by performing pure-cursor movements — i.e., repeated ESC-A commands or, for AUTOSTART ROM users, repeated K commands after ESC has been pressed. The need to perform these pure-cursor movements is annoying and inconvenient to say the least.

```
                        **********************************
                        *                                *
                        *   ZOOM AND SQUEEZE PROGRAM     *
                        *       BY CARY LITTLE           *
                        *     #101-2044 W. 3RD AVE.      *
                        *       VANCOUVER, B.C.          *
                        *       CANADA  V6J 1L5          *
                        *       JANUARY 1980             *
                        *                                *
                        * ENTER '300G3D0G' TO ACTIVATE   *
                        * (OR BRUN FROM DISK).           *
                        *                                *
                        * ENTER CTRL-Z TO ZOOM THE       *
                        * CURSOR TO THE RIGHT-MOST       *
                        * POSITION OF THE LINE (TEXT IS  *
                        * AUTOMATICALLY COPIED OVER).    *
                        *                                *
                        * ENTER CTRL-Q TO SQUEEZE THE    *
                        * COLUMN WIDTH TO 33.            *
                        *                                *
                        **********************************
                        WIDTH   EQU  $21        WINDOW WIDTH
                        CH      EQU  $24        HORIZONTAL CURSOR POSITION
                        BASL    EQU  $28        SCREEN BASE ADDRESS POINTER
                        KSWL    EQU  $38        INPUT HOOK (LO)
                        IN      EQU  $200       INPUT BUFFER
                        KEYIN   EQU  $FD1B      KEYPRESS ROUTINE
                                ORG  $300
0300: A9 09                     LDA  #<INHK     SET INPUT HOOK
0302: 85 38                     STA  KSWL        TO $INHK
0304: A9 03                     LDA  #>INHK
0306: 85 39                     STA  KSWL+1
0308: 60                        RTS
0309: 20 1B FD          INHK    JSR  KEYIN      GET A CHARACTER
030C: C9 91                     CMP  #$91       CTRL-Q PRESSED?
030E: D0 07                     BNE  CTRLZ      IF NOT, CHECK FOR CTRL-Z
0310: A9 21                     LDA  #$21       CHANGE WINDOW WIDTH
0312: 85 21                     STA  WIDTH        TO 33
0314: A9 A0                     LDA  #$A0       OUTPUT A SPACE
0316: 60                        RTS
0317: C9 9A             CTRLZ   CMP  #$9A       CTRL-Z PRESSED?
0319: D0 1F                     BNE  RTS1       IF NOT, RETURN
031B: A4 24             LOOP    LDY  CH         TAKE A CHARACTER
031D: B1 28                     LDA  (BASL),Y     OFF VIDEO SCREEN
031F: 48                        PHA
0320: E6 24                     INC  CH
0322: E6 24                     INC  CH
0324: A5 24                     LDA  CH         IF CURSOR POSITION IS
0326: C5 21                     CMP  WIDTH        AT FAR RIGHT,
0328: B0 0B                     BCS  FIN          THEN FINISHED
032A: C6 24                     DEC  CH
032C: 68                        PLA            ;STORE CHARACTER
032D: 9D 00 02                  STA  IN,X         IN INPUT BUFFER
0330: E8                        INX
0331: D0 E8                     BNE  LOOP       GET ANOTHER CHARACTER OFF SCREEN
0333: CA                        DEX            ;BUFFER FULL,
0334: 60                        RTS            ;  SO RETURN
0335: 68                FIN     PLA
0336: C6 24                     DEC  CH         SET PROPER CHARACTER
0338: C6 24                     DEC  CH           POSITION AND
033A: 60                RTS1    RTS            ;  RETURN
```

This problem can be avoided if the window width is 'squeezed' to 33 columns before LISTing the line and editing it. If this is done, the embedded blanks disappear and the line can be edited without worrying about the need to perform pure-cursor movements.

The window width can be changed to 33 be entering the command POKE 33,33 from BASIC immediate-execution mode. However, with the ZOOM and SQUEEZE routine in effect all that need be done is to press CTRL-Q. The width can be returned to its default value of 40 by simply entering the command TEXT from immediate-execution mode.

### How ZOOM AND SQUEEZE Works

ZOOM and SQUEEZE can be activated by BRUNning it from disk or by loading it, entering the command 300G from the monitor, and then returning to BASIC. The routine resides from $300 to $33A.

After it has been activated, the APPLE's input hook at $38 (low), $39 (high) is set equal to the ZOOM and SQUEEZE entry point at $309. Thereafter, all keyboard input is checked to see whether CTRL-Q or CTRL-Z has been pressed; if not, then nothing special happens.

If CTRL-Q is pressed, the short subroutine beginning at $310 and ending at $316 is executed. All this subroutine does is store $21 (decimal 33) at location $21 — this is the location in the monitor that contains the current window width. A blank is then displayed on the screen to indicated that this has occurred.

If CTRL-Z is pressed, the subroutine beginning at $317 is executed. What happens then is that the characters displayed on the screen from the current cursor position to the end of the line are placed in the input buffer one-by-one. If the butter is overflowed, the program line will be backslashed and cancelled in the ordinary way.

Details of the programming algorithms involved can be easily deduced by inspecting the accompanying source listing tor ZOOM and SQUEEZE.

NOTE: It is possible to speed up the repeat-key tunction by soldering a 100K resistor in parallel to the resistor at position R4 on the APPLE keyboard unit. For details, see the article 'REPEAT KEY SPEED-UP' by V.R. Little in the February 1980 edition ot APPLEGRAM, the newsletter of the Apples British Columbia Computer Society, Vancouver, B.C.

μ

*Gary B. Little first became interested in computers by writing data analysis programs in FORTRAN on an IBM 370/168 for an M. SC. degree in Physical Chemistry (Microwave Spectroscopy). Ultimately he became interested in microcomputing and purchased an APPLE II micro 1½ years ago.*

*He was past president of APPLES BRITISH COLUMBIA COMPUTER SOCIETY, an an APPLE user group located in Vancouver, B.C. Gary is currently the treasurer of this group.*

# EDUCATIONAL SOFTWARE
## for the APPLE II and APPLE II PLUS

### VERBAL SKILLS

Diskettes with Applesoft Programs and data base can be used to improve verbal skills interactively. Intended as study aids for college board type exams (E.G., SAT, GRE, LSAT, ACT, MAT, ETC). Programs include a resident editor for expanding or modifying data lists. Realistic multiple choice questions are generated with several options. Requires no computer experience making it ideal for students.

| | |
|---|---|
| **Vocabulary Builder-** | Over 1600 entries with option of matching synonyms or antonyms - $19.95 |
| **Analogy-** | Over 1200 word relationships - $19.95 |
| **Sentence Completion-** | Equivalent to 6 SAT tests -$19.95 |
| **Verbal Skills Pak-** | All three diskettes for just $44.95 |

**SLIWA ENTERPRISES**
EDUCATIONAL SOFTWARE DEPARTMENT
POST OFFICE BOX 7266
HAMPTON, VIRGINIA 23666

*Apple is a registered trademark of Apple Computer, Inc.

---

**THE CORRESPONDENT** is sure to be one of the most versatile programs in your library! It can be used as:

**A TEXT PROCESSOR:** Upper/lower case, 1-80 col. s (4-way scrolling!), text move/copy/insert/delete, tabbing, justify text, auto centering and more!

**A DATABASE:** (with or without printer!) Extremely fast find routine & easy editing make it a natural for free-form data files. Create & fill out forms, access phone lists or index your magazines.

**A PROGRAMMING UTILITY:** (printer or not) Examine, edit, transfer random or sequential text files. Create versatile exec files. Even put bi-directional scrolling in your own programs!

**SEE YOUR LOCAL DEALER FOR A DEMO TODAY!**

**PRICE: $34.95**
California Residents add 6% Sales Tax
**DISKETTE ONLY:** 32K FP ROM, 48K RAM

**the correspondent**
By Roger Wagner

WRITE OR CALL FOR MORE INFORMATION
**southwestern data systems**
P.O. BOX 582-M • SANTEE, CA 92071 • 714/562-3670

---

# HAS YOUR APPLE READ ANY GOOD PROGRAMS LATELY?

## APPLE II DISK SOFTWARE

### DATA BASE MANAGER
### IFO PROGRAM

The IFO (INFORMATION FILE ORGANIZER) can be used for many applications such as sales activity, check registers, balance sheets, client/patient records, labratory data reduction, perscription information, grade records, mailing lists, A/R, job costing and much more. This can be accomplished easily and quickly without prior programming knowledge.
Up to 1000 records with a maximum of 20 headers (catagories) and 10 report formats (user defined) can be stored on a single diskette, information can be sorted on any header, both ascending and descending in alpha/numeric field. Mathematical functions can be performed on any 2 fields to manipulate the information. Information can be searched on any header using >, <, = >, = <, =, and first letter. Mailing list format provided. Fast assembly language sort, search and read routines. Many error protection devices provided. Put your application program together in minutes instead of hours.

**PROGRAM DISKETTE and instruction manual....$100.00**

**MAILING LIST PROGRAM...$40.00**

### INVENTORY PROGRAM

2 disk drives, menu-driven program. Inventory categories include: STOCK#, DESCRIPTION, VENDOR ID, CLASS, LOCATION, REORDER PT., REORDER QTY, QTY ON HAND. All records can be entered, changed, updated, deleted, or viewed. Reports can be sorted in ascending/decending order by any category. There are 7 search reports (3 automatic). Calculates $ VALUE of inventory and YTD, MTD, and period items sold, Accumulates inventory over a 13-month period. Plus much more. Requires a 132-column, serial/parallel printer, Complete turnkey operation with bootstrap diskette.
Program diskett and instruction manual...$140.00

### PAYROLL PACKAGE

2 disk drives, menu-driven program. Employee history include: NAME, ADDRESS #, ADDRESS #2 CITY, STATE, ZIP, FED EX, STATE EX., SOCIAL SEC.#,DATE EMPLOYED, DEPT #, CODE, EMPLOYEE #, STATUS, MARITAL STATUS, PAY RATE, OT RATE, VAC RATE, # VAC HRS. and PENSION PLAN. Program can generate weekly or biweekly payroll. Prints W-2, QTR REPORT, PAY CHECKS, MASTER AND CURRENT files. FEDERAL and STATE witholding taxes are built into program. Maintains a CASH DISBURSEMENT journal, Accumulates payroll for a 53 week period. Generates numerous type of payroll reports. Allows data to be searched, sorted and edited. Prints DEDUCTION register and more. Maintain up to 125 EMPLOYEES/EXPENSES for quick and easy PAYROLL. Numerous error protection devices provided.
PROGRAM diskette and instruction manual...$240.00
*PLEASE SPECIFY STATE WHEN ORDERING

### APARTMENT MANAGER

2 disk drive, menu driven program written in assembly language and APPLESOFT II. All you will ever need to manage your apartment. Handles up to 6 BUILDINGS with a maximum of 120 units each. Complete turnkey operation. Data categories include APT #, TYPE, TENANT NAME, PETS, CHILDREN, SECURITY DEP., PET DEPOSIT, POOL DEP, MISC DEP, RENT ALLOWANCES, DATE MOVED IN, VACANCY DATE, REFFERAL, CONDITION OF APT, DAMAGE AMT and COMMENT LINE. Search, sort, enter, edit and vacate tenates. Maintains a MTD and YTD rent recipts as well as complete utility reports, rent lost by vacancies. Maintains expenses, vacated tenants report and much more.
PROGRAM DISKETTE and INSTRUCTION MANUAL...$350.00

### PROFESSIONAL TIME AND BILLING

2 disk drive program written in assembly language and APPLESOFT II. Completely menu-driven, Maintain all billing of clients and personnel. Generates and invoices. Numerous reports based on all types of criteria. Easy data entry for RATES, CLIENTS, and MATTERS. Has SEARCH, SORT, CHANGE (on screen editing), VIEW and BALANCE FORWARD. IF your are a JOB CONTRACTOR, ATTORNEY, ACCOUNTANT, GENERAL CONSULTANT, or anyone that needs to charge for time, this program is a must. Complete turnkey operation. Numerous REPORTS are produced to aid in the TIME ANALYSIS PROCESS. All this and much more.
PROGRAM DISKETTE and INSTRUCTION MANUAL...$325.00

### SPEED READING

**PROGRAM DISKETTE AND INSTRUCTION MANUAL...$100.00**

ALL PROGRAMS REQUIRE 48K and APPLESOFT II ON ROM OR AND APPLE II PLUS. ALL SOFTWARE IS COMPATABLE WITH PASCAL SYSTEMS. PROGRAMS RUN FROM ANY PORT OF THE COMPUTER WITH SERIAL/PARALLEL PRINTERS. REQUIRES 1 DISK DRIVE UNLESS OTHERWISE NOTED.
SEND CHECK/MONEY ORDER or C.O.D. To:

**SOFTWARE TECHNOLOGY for COMPUTERS**
**P.O BOX 428**
**BELMONT, MA 02178**

OR AVAILABLE FROM YOUR LOCAL DEALER.

Welcome to the second issue of the Ohio Scientific Small Systems Journal in Micro.

In this issue, Ohio Scientific is pleased to introduce a new concept in computer interfacing — the Sixteen Pin I/O BUS. The BUS concept as well as several boards and applications are covered in the following pages.

Also in this issue, a short, graphics oriented game in BASIC called 'FOO' is presented.

Reader suggestions on article content are welcome. Please submit them to:
Ohio Scientific, Inc.
Small Systems Journal
1333 S. Chillicothe Rd.
Aurora, Ohio 44202

**The Ohio Scientific Sixteen Pin I/O BUS**

Ohio Scientific is pleased to introduce a unique new product line — The 16 Pin I/O BUS. With this system it is possible to add up to eight special function boards while occupying only one backplane slot.

This is made possible by a novel BUS extension method which allows decoding, timing and eight bits of data to be carried on standard, inexpensive 16 pin ribbon cables.

Up to eight inexpensive 16 pin cables with standard DIP connectors may be attached to a single CA-20 board which in turn occupies one slot of the standard Challenger backplane. Alternately, one 16 pin I/O BUS cable may be attached to the CA-15 board at the rear of all C4P and C8P products. Note, in the case of the C4P-MF this allows system expansion beyond the normal four slot backplane.

Currently, five HEAD END CARDS are available for interconnection to the system via the CA-20 or CA-15 boards.

**Computer Interface to Sixteen Pin I/O BUS**

The 16 pin I/O BUS may be attached to your computer via two different boards — the CA-15 or the CA-20. The descriptions of these boards are as follows:

**CA-15 Board**

The CA-15 board is a standard accessory interface installed on the following Ohio Scientific systems: C4P-MF, C4P-DMF, and C8P-DF.

The CA-15 is mounted at the rear of the computer and contains the following interface connections:
Joystick and numeric keypad
Modem and serial printer
Sixteen PIA lines (normally used for the Home Security system — AC-17P)
Sixteen Pin I/O BUS

The interconnect for the Sixteen Pin I/O BUS is simply a 16 pin DIP socket. To use the BUS, all that you have to do is attach one end of the 16 pin ribbon cable to the CA-15 board and the other end of the cable to one of the HEAD END CARDS.

Please note that some of the HEAD END CARDS require more power than may be practically carried via the ribbon cable alone. Therefore, some of the cards require auxiliary power supplies.

**CA-20 Board**

The CA-20 board contains all the necessary logic to decode eight distinct HEAD END CARD interfaces. The actual interconnect, as with the CA-15, is via simple 16 pin DIP sockets and standard 16 pin ribbon cables.

The CA-20 board also requires one slot of your computer's backplane. But remember, from this one slot you gain access to a maximum of eight accessory boards.

The CA-20 is recommended for use in the Ohio Scientific C2 series and C3 series computers. It can also be installed in C4P and C8P series systems with some modification to the CA-15 interface.

Since the logic required for the I/O BUS interface is pretty simple, an additional feature was added to the CA-20 board — a crystal controlled 'time-of-day' clock (hardware) subsystem. The operation of the clock, excepting reading time and setting time, is totally independent of the host computer. As a matter of fact, with the included on-board, auto-recharging, battery back-up, your computer may actually be turned off for several months without losing time.

The features of the clock subsystem are as follows:

Hours, minutes, seconds and 1/10 seconds
Day of week
Day of month
Month of year
Four Year calendar

If you happen to own (or use) a C2 series or C3 series computer, the CA-20 board can actually control the power cycling of the entire computer when equipped with an optional power sequencer package. This means you can preset a time (month, day, hour, etc.) within the clock subsystem and that preset time agrees with the actual time, A.C. power is applied to the entire computer system through the power sequencer. At a later time, the system's A.C. power may also be removed and the system shut down under software/clock subsystem control.

For applications where the clock subsystem is not required, the CA-20A will perform all the Sixteen Pin I/O BUS functions associated with full-feature CA-20.

**HEAD END CARDS**

HEAD END CARDS is a general name used to describe any or all of the special function boards which attach to the Ohio Scientific Sixteen Pin I/O BUS. There are currently five such boards and, with the exception of the CA-22, they will only interface with the computer via the Sixteen Pin I/O BUS.

Please note, as detailed earlier, you must use a CA-15 or a CA-20 board at the 'computer end' of the Sixteen Pin I/O BUS to complete the interface.

In the following pages, a brief product and applicatior

description of the currently available HEAD END CARDS will be presented.

**Bit Switching and Sensing — The CA-21**

The CA-21 is a 48 line parallel I/O board featuring three 6821 PIAs (peripheral interface adapters) and prototyping/interconnect areas.

The use of PIAs in the design allows for maximum interface versatility as you may configure any one of the 48 I/O lines as either an input or an output. As outputs, each line is capable of driving a minimum of one standard TTL load.

Additional versatility is added because 24 of the lines, when configured as outputs, may simultaneously function as inputs. This feature, although somewhat confusing, is extremely useful for applications such as switch matrix decoding.

Each of the 48 lines is brought out to two foil pads (suitable for wire wrap stakes) as well as a location on one of four 12 pin Molex-type female edge connectors. There are also eight 16 pin DIP socket locations which are intended for use as prototyping areas. Additionally, the 12 PIA 'hand-shaking' lines are brought to 12 single foil pads.

The CA-21, with proper buffering, may be used for virtually any computer controlled bit switching or bit sensing application that you can imagine. With a full complement of eight CA-21s interfaced via the CA-20, a total of 384 individually controllable I/O lines are possible!

An interesting application using one CA-21 board would be a complete, is somewhat slow, emulation of the standard Ohio Scientific BUS.

A more standard application might be augmenting the standard Home Security System (AC-17P) with 'hard-wired' sensors.

One type of sensor you could easily add is a standard windor 'perimeter detector'. This could be done with commercially available adhesive foil tape. You could then detect a break-in (through a broken window) by sensing a break in the foil tape.

Another useful application you could set up in concert with the AC-12P wireless A.C. Remote Control, might be sensing when a room is entered. You could accomplish this with pressure-switch door mats or door switches. When room entry is detected, the lights could be turned on or, turned off on exit.

If you are designing any sort of dedicated control system, the CA-21 is an ideal choice. You can easily sense innumerable types of input (pressure transducers, flow sensors, switches, etc.) while controlling outputs from a simple single LED display to a network of solid state relays controlling A.C. power.

**EPROM Programmer — The CA-23**

The CA-23 is an EPROM programmer designed for use with the growing families of 5 volt only EPROMS. With the CA-23 you can program and verify all 1K through 8K byte EPROMS of this type. Note these parts are often identified as 8K — 64K bit EPROMS.

The CA-23 can program (or verify) data in two basic modes — EPROM to/from EPROM or EPROM to/from computer RAM memory. Additionally, EPROM data may be read directly into the computer's RAM memory.

There are four LED indicators on the CA-23. The first is 'SOCKET UNSAFE'. This means that a programming voltage is present at the socket and if you insert or remove an EPROM it is likely to be damaged.

The second indicator is 'PROGRAMMING'. This means that your EPROM is currently being programmed.

The third indicator is 'ERROR'. This means that somewhere along the line your programming attempt was unsuccessful.

The final indicator is 'PROGRAM COMPLETE'. This means that your program and verification was successful.

The most intriguing application for this product is the creation of 'custom' parts for your computer or peripherals. This could range from a new system monitor to a new high level language. It could even include a new character generator for your CRT or printer. Note, however, tinkering around with the internals of computers and peripherals requires a fairly high degree of technical expertise. Also, most manufacturer's warranties are voided by these types of modifications.

Several OEM (original equipment manufacture) and Research/Development applications will be immediately obvious to those you involved in that work.

The CA-23, as previously mentioned, is designed for use with 1K through 8K byte EPROMS. These parts come in various package styles and have various product names. For example, Intel's 2Kx8 part is the 2716, Texas Instruments' part is known as the 2516.

The CA-23 has both 24 pin and 28 pin zero insertion force sockets for reading, programming and verifying the EPROMS.

**Prototyping — The CA-24**

The CA-24 is a solderless bread-board designed for proto-typing, experimental and educational applications.

The bread-boarding is made up of seven solderless plug-strips of the type manufactured by AP Products. Two of the plug-strips contain a connection matrix of 5 by 54 connecions and are used as signal distribution points. Another pair of 96 location plug-strips are for powering the bread-board area. The actual experimenter area is comprised of three plug-strips, each with a 10 by 64 location connection matrix. Additionally, sixteen LED indicators and sixteen DIP switch positions are provided for signal observation and control functions.

Board I/O is via TTL latches and bi-directional PIA ports as well as direct (buffered) data, signal and control lines from the computer BUS. This method allows you to directly interconnect devices such as 6850 ACIAs in addition to doing more 'isolated' and/or independent circuits.

The CA-24 also contains a 'clock' generator which is continously variable from approximately 25,000 Hz. through 70,000 Hz. You may also connect the clock to an on-board 16 stage divider chain. This allows division of the fundamental frequency by as little as $2^1$ (2) to as much as $2^{16}$ (65,536).

The applications for the CA-24 are primarily prototyping and experimenting. Parts may be inserted and removed from the terminal strip blocks over and over. Interconnection of parts is accomplished simply with solid, narrow guage wire jumpers. Errors in design or connection are extremely easy to correct.

The CA-24 lends itself very well to structured experiments that are common in the educational environment. It is an ideal tool to aid in the teaching of computer and computer interface fundamentals.

**Accessory Interface — The CA-25**

The CA-25 is designed to implement some of the functions normally associated with the CA-15 interface board.

It allows you to directly connect the Home Security System (AC-17P) and/or the Wireless A.C. Remote Control System (AC-12P) to C2 and C3 series computers. Additionally, if you own an older Ohio Scientific computer, you can now easily connect these systems to it.

An extremely useful application of the CA-25 is associated with small business systems. Using the CA-25 with the Home Security System, and perhaps a CA-15V (Universal Telephone Interface with speech synthesizer output), the computer could do payroll, inventory, etc. by day and 'guard' the shop by night.

**Analog I/O — The CA-22**

The CA-22 is a high speed analog I/O module. Although the CA-22 is classified as a HEAD END CARD, it differs from the rest of the family in that it may also be plugged directly into the computer's standard internal BUS. This allows for maximum flexibility in the use of the CA-22.

The analog input section of the CA-22 consists of a 16 channel analog multiplexer. This means that you may connect up to 16 separate signals directly to the CA-22. Also included is a sample and hold circuit followed by the analog to digital converter circuitry.

The A to D converter is capable of either 8 bit or 12 bit operation. You may select these options under software control.

The accuracy of the converter is plus or minus one in the least significant bit. The stability of the circuit is rated at one millivolt drift per degree Centigrade.

The A to D conversion is extremely fast. It is capable of digitizing up to 66,000 samples per second in the 8 bit conversion mode and 28,000 samples per second in the 12 bit mode. Shannon Sampling Theory states that signals should be sampled at twice their frequency. Therefore, it is possible for you to convert signals with a frequency greater than 30K Hz. Clearly, high fidelity audio is well within the spectrum of the CA-22.

The multiplexer has very high impedance inputs and is capable of accepting inputs in the range of -10 volts through +10 volts. The input is jumper selectable for other settings including a single sided range of 0 through +10 volts.

Due to the indeterminable nature of the actual inputs that you may actually apply to the CA-22, only the multiplexer inputs are brought out. However, a quad op-amp is laid out in foil which you may populate in several different modes to handle some of the more 'common' input configurations.

The analog output section of the CA-22 consists of two identical high speed digital to analog converters. Each DAC can convert either 8 bits or 12 bits of data. Data input to the DACs is latched in such a manner that, when in the 8 bit conversion mode, the other four (of the total of twelve) bits are continuously output at a predefined value. You may, of course, define that value under software control.

The output of each DAC is buffered with a high speed op-amp capable of changing 20 volts every microsecond. The standard configuration of each output is bi-polar with a voltage swing of -10 volts through +10 volts. This is jumper selectable to allow a uni-polar output of 0 through +10 volts.

Some additional I/O capacity is provided on the CA-22. There are three TTL level inputs and six open collector logic outputs. These are strappable to be either standard TTL level outputs or high-voltage outputs.

You can use the CA-22 for a multitude of analog sensing and/or analog controlling applications.

Using the proper transducers and the 16 input channels, you can monitor the temperature in several zones of a home or office. By extending this system with a CA-21, you could maintain precise temperatures by switching the proper controls on and off.

Another interesting, if somewhat obvious application, is in audio processing. Reverberation, phase shifting and echoing are just a few of the uses you could implement.

If you used blocks of RAM for data storage, other applications such as frequency doubling, etc., could be experimented with.

If you apply more sophisticated software techniques, such as a fast Fourier transform, on stored input data, very elaborate signal processing becomes realizable. Projects such as audio spectrum analyzers and speech recognition experiments are certainly practical. Note, in these types of applications you are likely to find some signal pre-processing in hardware is certainly beneficial - if not totally necessary.

If you employ both DAC outputs and the on-board unblanking circuit, X-Y oscilloscope plotting is an interesting application. By using these techniques and one or more of the analog inputs, you can construct a digital storage scope. Note, both of these applications require that you have access to an oscilloscope capable of X-Y input as well as blanking.

**Summary**

With the introduction of the 16 pin I/O BUS, Ohio Scientific has opened a new world on interfacing capabilities for both the large and the small computer user.

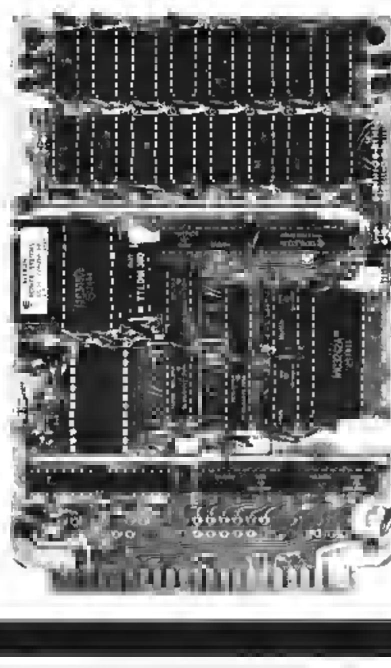
Systems ranging from totally automated sampling and control stations to complete R/D setups to educational lab stations are now available to you via standard building blocks and standard computer systems.

For pricing and availability, contact your nearest Ohio Scientific dealer.

## FOO

This is an amusing graphics game that simulates a twisting road scrolling up from the bottom of the screen. You must avoid going off the road. Speed and road width are selectable. Pedestrians are also optional, with a bizarre twist. At your option pedestrians are to be avoided or run down for points. FOO runs on disk based C4P and C8P video systems. The tone generator is used to provide sound. The program is easily adapted to OSI BASIC-in-ROM computers.

```
100  POKE 2893,55:POKE2894,8:POKE2073,96
110  BS=55040:SM=2:MS=1:KY=57088:ME=54144+15:MI=0:RN=0
115  ML%=0
117  SN=255
120  LP=5
130  PL=2/LP
135  POKE57089,1
140  POKE9680,32:POKE56832,2
150  C=226
155  KP=0
160  IFA$='Y'THENME=EM:WI=WF:GU=UG:GOTO270
170  FORI=1TO30:PRINT:NEXTI
180  PRINT'F O O'
190  PRINT:PRINT'  R A C E W A Y'
200  PRINT:PRINT'You run at your own risk!'
210  PRINT:PRINT'<== LEFT=1 RIGHT=2 ==>'
215  PRINT:PRINT'OVERDRIVE=RUBOUT'
220  PRINT:PRINT'SUGGEST WIDTH=20, DELAY=20'
230  PRINT:INPUT'INITIAL WIDTH (0-30)':WI
240  PRINT:INPUT'DELAY (1-20)';ME:EM=ME
245  PRINT
250  GU=0:INPUT'PEDESTRIANS
     (Y/N)';X$:IFLEFT$(X$,1)='Y'THENGU=.3
255  UG=GU:PRINT
257  IFGU=0THEN270
260  KP=0:INPUT'KILLER FOO
     (Y/N)';X$:IFLEFT$(X$,1)='Y'THENPK=1
270  PRINT:PRINT'Hidden wonders await
     the':PRINT'Masters!'
280  FORI=1TO30:PRINT:NEXTI
290  WD=WI:WF=WI:ME=55104+15-ME*64:WT=(30-WI)/2
295  IFA$='Y'THENRETURN
300  FORM=1TOLP:GOSUB600:GOSUB500:ML%=ML%+1:NEXTM
350  WI=WI-1
360  LP=LP*1.14
370  IFWI>4THEN300
380  SM=SM+.2:MS=MS+.1
400  FORM=1TOLP:GOSUB600:GOSUB500:ML%=ML%+1:NEXTM
450  WI=WI+1
460  LP=LP*.85
470  IFWI<WDTHEN400
475  IFWD<2THENWD=WF
480  WD=WD*.75
490  GOTO300
499  REM OUTPUT A FRAME
500  RN=RN+SM*RND(1)-MS
510  WT=WT+SGN(RN)
520  IFWT+WI>28THENWT=WT-1:RN=0:GOTO520
530  IFWT<0THENWT=0:RN=0
540  IFWI>8ANDRND(1)<GUTHENPOKEBS+WT+1+INT
     (WI*RND(1)),240
550  PRINTSPC(WT);'><';SPC(WI);'><'
560  RETURN
599  REM MOVE BALL
600  POKEKY,128:K=PEEK(KY):KK=0:POKEKY,64:K2=PEEK(KY)
610  IFKAND128THENME=ME-1:KK=-1+0*RND(1)
620  IFKAND64THENME=ME+1:KK=1
630  IFK2AND4THENME=ME+KK
640  IFPEEK(ME)<>32THEN700
650  POKEME,G
660  RETURN
700  GY=PEEK(ME):IFGY=240ANDPKTHENKP=
     KP+1:GOSUB2000:GOTO650
710  POKE 2073,173
715  FORI=100TO250STEP5:POKE57089,I:NEXTI
719  POKE57089,1
720  PRINT'YOU BLEW IT!!!'
725  PRINT
730  MI=ML%*PL
750  PRINT'AFTER ';MI;' MILES'
755  IFPKTHENPRINT'AND ';KP;' KILLS'
757  P R I N T : P R I N T ' T O T A L
     POINTS:';INT(MI+4*(1-PK)*MI+100*KP)
760  GOSUB1000
770  K=1
780  FORI=1TO1000*K:NEXTI
790  IFPEEK(KY)<>1THEN790
800  POKE9680,95
805  POKE57089,1
810  GOTO5000
1000 IFPKTHENWD=KP:GOTO1030
1010 WD=MI/WF
1030 PRINT:PRINT'Congratulations!'
1040 PRINT'You may now call yourself'
1050 PRINT:PRINT'        ';
1060 IFWD<3THENPRINT'LITTLE';:GOTO1200
1070 IFWD<5THENPRINT'TENDER';:GOTO1200
1080 IFWD<12.5THENPRINT'MEDIOCRE';:GOTO1200
1090 IFWD<25THENPRINT'BIG';:GOTO1200
1100 IFWD<38THENPRINT'MASTER';:GOTO1200
1110 IFWD<50THENPRINT'GRAND';:GOTO1200
1120 PRINT'CHEATER';
1200 PRINT' FOO';
1210 IFGY=240THENPRINT' KILLER';
1220 PRINT'!'
1230 RETURN
2000 SN=SN-5
2003 IFSN=50THENSN=255
2005 POKE57089,SN
2010 POKE 57089,1
2020 RETURN
5000 INPUT'AGAIN';A$:A$=LEFT$(A$,1)
5010 IFA$<>'Y'THEN6000
5020 INPUT'SAME';A$:A$=LEFT$(A$,1)
5025 IFA$<>'Y'THENCLEAR
5030 GOTO100
6000 END
```



# VIZA — KIM

**A KIM Monitor extension progrem which provides the eutomatic displey of the important system peremeters et eech step. The discussion reveals some deteils of the 6502 interrupt hendling mechanism.**

Joel Swank

After reading George Lang's article on his U-PANEL project (MICROCOMPUTING, January 1979), I decided to implement his idea on my KIM-1 system. U-PANEL is a front panel display for KIM. It uses an extension of the KIM single step circuit (SST) and a small routine to dump the processor registers in binary to a panel of discrete LEDs. This is done by connecting the KIM SST signal on pin E-17 to the IRQ Interrupt line on pin E-44. The SST signal is generated every time the CPU SYNC signal is generated and the instruction being executed is not located in the KIM ROM. SYNC is generated with each opcode fetch. Normally during KIM single step operation the SST signal is switched to the nonmaskable interrupt (NMI) line. This causes an interrupt during the first cycle of each instruction. Since an instruction cannot be interrupted in the middle, the interrupt is recognized immediately after the instruction is finished. The NMI vector cannot be set to a routine outside the KIM ROM while the SST switch is on because the first instruction of that routine will also cause the NMI interrupt to be taken, resulting in a continuous loop. Instead of the NMI George switched the SST signal to the IRQ line, KIMs maskable interrupt. This allows the interrupt to be vectored to any routine anywhere in the system rather than just the KIM ROM. The IRQ vector was changed to the register dump routine which returns control to KIM after outputtine the registers to U-PANEL. This routine must run with interrupts disabled to prevent it from being interrupted.

Since I don't particularly care for reading binary lights, I decided to dump in HEX to my CRT terminal. This saves building the U-PANEL. and provides a more readable display. The changes to George's program were simple and I soon had my code ready to test, but I couldn't get it to work properly. I double checked everything and it all looked OK. So I started to analyze the problem.

The register dump to the CRT was working, but the CPU was not being interrupted after each instruction. It would execute a few insturctions and then stop. When I pushed GO it would execute a few more and stop. After a little thought I decided to see which instructions were being executed without being interrupted. Soon a pattern emerged. The CPU was being interrupted only after instructions whose execution time were two cycles. Any instruction whose execution time was 3 or more cycles was not being interrupted. Why? The answer lies in the MOS Technology hardware manual. The NMI Interrupt is edge sensitive. That is, the Interrupt is recognized by the change from high to low not just the presence of the low signal. Also, once the transition has occured the processor will be interrupted before the next Instruction starts, no matter what. The IRQ is not edge sensitive. A low on the IRQ line must coincide with a zero in the processor interrupt flag and the last cycle of an instruction. If the IRQ line goes low and high again while the CPU is not ready to accept inter-

rupts, the interrupt will be missed. In this case the SST signal because it is driven by SYNC will be low during the first cycle of an instruction and because of propagation delay, slightly into the second cycle. Therefore any instruction that is 3 cycles or longer will cause the interrupt to be missed. So the interrupt occurs only after two cycte instructions (the 6502 has no one cycle instructions).

To fix this problem the SST pulse must be lengthened to last at least as long as the 6502's longest instruction. The circuit in figure one does this. It uses a one shot to extend the putse. This circuit produces a pulse of about one millisecond, much longer than needed, but it doesn't matter as long as the pulse is long enough. This circuit will provide a properly operating U-PANEL.

After resolving the pulse length problem I decided to add a slow motion feature. This woutd be a mode that would execute an instruction and then, after dumping the registers, instead ot returning to KIM, would delay for a programmable amount ot time and execute the next instruction. This would allow the execution ot a program in slow motion without pushing GO between each instruction. The code needed to add this feature is fairly simple and it was soon ready to test. I implemented it with a time value at $E9. This value is the delay time in in quarter seconds. Zero means slow motion not in etfect. On first try I set the delay to two seconds and started the program. The first instruction was executed and the registers dumped, but there progress stopped. The delay was working properly and the display being updated every two seconds but the PC was not advancing. It was stuck on the second instruction. I stopped execution and started it again. This time the second instruction was executed and it stuck on the third. Once again the problem was in the non edge sensitive IRQ interrupt.

When in normal mode, each instruction in the dump routine generates a putse. These pulses are ignored during execution of the dump routine because it runs disabled. The pulses stop once execution enters the KIM ROM. The RTI instruction that KIM executes as a result of pushing GO enables the IRQ and the first instruction in the object program generates a pulse that causes an interrupt immediately after it executes. The dump routine is then executed, and control is returned to KIM to wait for the next GO. In slow motion mode the GO routine is executed via a JMP instruction trom the dump routine. If the pulse generated is longer than the time needed to execute the GO routine (about 38 microseconds) the IRQ line will still be low from the JMP instruction when the RTI instruction is executed. This will cause an interrupt immediately after the RTI instruction and no instruction of the object program will be executed. To solve this problem, the pulse must be shortened to less than the duration of the GO routine. This can be done by changing the resistor in figure one to 2K Ohms. This generates about a 35 microsecond pulse, longer than the longest 6502 instruction but shorter than the KIM GO routine.

I called my version of the program VIZA-KIM. The code for version 1 is included. It provides a formatted display on the CRT after each instruction is executed. Version 2 has been enhanced to display in large characters on my SWTPC GT-6144 graphics board. This display on my 19 inch TV can be read by an entire room of people. VIZA-KIM makes an execellent device for learning the operation of the CPU. The exact effect of each instruction can be seen.

The VIZA-KIM dump displays the program counter (PC) and the first three bytes of data at that location. A nice enhancement would be to include a line for a disassembled instruction. The next line is for the stack pointer (SP). The current stack pointer is displayed along with three bytes from the stack page. The tirst byte is where the next push operation will store its data. The 6502 stack pointer always points to the next available byte. The next two bytes are the data from the last two push operations, or the data that will be read by the next two pull operations. If the last push operation was a jump subroutine (JSR) instruction this will be the return address minus 1. Next are the index registers (X and Y) and the accumulator (A). Last is the processor status register (P). All data is displayed in HEX except for P. P is tormatted in binary since its individual bits have separate meanings.

To use VIZA-KIM set the IRQ vector ($17FE) to the address of the dump routine and turn on the new SST switch. Be sure the use P register at location $F1 has the interrupt flag (bit 2) set to zero, since the object program must run with interrupts enabled. To use slow motion mode set $E9 to the number ot quarter seconds of delay desired, enter the address of the object program and press GO. Instructions will be executed one at a time after the desired delay. To stop execution hold down any key on the KIM keyboard. To use normal mode clear $E9 to zero and enter the address of the object program. Operation will be the same as in KIM SST mode.

VIZA-KIM makes one aware of each change of the state of the processor as each instruction is executed. This makes bugs more easily spotted as well as giving one a better understanding of how the 6502 works.

μ

```
   VIZA-KIM

    PC    DATA
   2008 C01A90
  SP=FF 305748
   X=06 Y=0A
      A=00
        P
    NV BDIZC
    00100000

    PC    DATA
   200A 90F885
  SP=FF 305748
   X=06 Y=0A
      A=00
        P
    NV BDIZC
    10100000

    PC    DATA
   2004 998000
  SP=FF 305748
   X=06 Y=0A
      A=00
        P
    NV BDIZC
    10100000
```

```
                  VIZA-KIM VERSION 1 SEPTEMBER 1979

                  TTY VERSION OF VIZA-KIM

                  EXTENDED MONITOR FOR THE KIM-1

                  ZERO PAGE STORAGE

2000              SLOMO   *     $00F9  DELAY TIME
2000              MSGPTH  *     $00EA  POINTER FOR STRING
2000              MSGPTL  *     $00EA  PRINT ROUTINE
2000              MSGPTH  *     $00EB
2000              PPLO    *     $00EC  STACK POINTER
2000              PPHI    *     $00FD
2000              SAVY    *     $00EE

                  PROCESSOR REGISTER SAVEAREA

2000              PCL     *     $00EF
2000              PCH     *     $00F0
2000              PREG    *     $00F1
2000              SPUSER  *     $00F2
2000              ACC     *     $00F3
2000              YREG    *     $00F4
2000              XREG    *     $00F5

2000              POINTL  *     $00FA
2000              POINTH  *     $00FB

                  EQUATES

2000              ZERO    *     $0000
2000              CR      *     $000D
2000              LF      *     $000A


                  LABELS

2000              CRLF    *     $1E2F  PRINT CRLF
2000              START   *     $1C4F  ENTRY TO KIM
2000              PRTPNT  *     $1E1E  PRINT $FA & $FB
2000              OUTCH   *     $1EA0  PRINT CHARACTER
2000              PRTBYT  *     $1E3B  PRINT ACCUM IN HEX
2000              INITS   *     $1E88  INIT I/O PORT
2000              TIMST   *     $1707  START TIMER
2000              TIMOUT  *     $1707  READ TIMER
2000              GOEXEC  *     $1DC8  KIM GO ROUTINE
2000              GETKEY  *     $1F6A  READ KIM KEYBOARD

3000              VIZA1   ORG   $3000

3000 85 F3        IRQ     STA   ACC     SAVE ACCUMULATOR
3002 68                   PLA
3003 85 F1                STA   PREG    SAVE STATUS REGISTER
3005 68                   PLA
3006 85 EF                STA   PCL     SAVE PROGRAM COUNTER
3008 85 FA                STA   POINTL
300A 68                   PLA
300B 85 F0                STA   PCH
300D 85 FB                STA   POINTH
300F 84 F4                STY   YREG    SAVE Y REISTER
3011 86 F5                STX   XREG    SAVE X REGISTER
3013 BA                   TSX
3014 86 F2                STX   SPUSER  SAVE STACK POINTER
3016 20 88 1E             JSR   INITS   INIT I/O
3019 A9 E8                LDAIM VMSG    PRINT HEADER
301B 85 EA                STA   MSGPTL
301D A9 30                LDAIM VMSG    /
301F 20 D7 30             JSR   MSGWRT
3022 20 1E 1E             JSR   PRTPNT  PRINT PC
3025 A9 20                LDAIM '
3027 20 A0 1E             JSR   OUTCH
302A A0 00                LDYIM $00     CLEAR INDEX
302C B1 FA        DALUP   LDAIY POINTL  PRINT 3 BYTES OF DATA
302E 20 CF 30             JSR   PSAVY   STARTING AT THE
3031 C8                   INY           PRORAM COUNTER
3032 C0 03                CPYIM $03
3034 D0 F6                BNE   DALUP
3036 A9 0C                LDAIM SPMSG   MORE HEADING
3038 85 EA                STA   MSGPTL
303A A9 31                LDAIM SPMSG   /
303C 20 D7 30             JSR   MSGWRT
303F A5 F2                LDA   SPUSER  LOAD STACK POINTER
3041 85 EC                STA   PPLO
3043 20 3B 1E             JSR   PRTBYT  PRINT SP VALUE
3046 A9 20                LDAIM '
3048 20 A0 1E             JSR   OUTCH
304B A9 01                LDAIM $01
304D 85 ED                STA   PPHI
304F A0 00                LDYIM $00     PRINT 3 BYTES FROM STACK
3051 B1 EC        STKLUP  LDAIY PPLO
3053 20 CF 30             JSR   PSAVY
3056 C8                   INY
3057 C0 03                CPYIM $03
3059 D0 F6                BNE   STKLUP
305B A9 12                LDAIM XMSG    MORE HEADING
305D 85 EA                STA   MSGPTL
305F A9 31                LDAIM XMSG    /
3061 20 D7 30             JSR   MSGWRT
3064 A5 F5                LDA   XREG
3066 20 3B 1E             JSR   PRTBYT  PRINT X REGISTER
3069 A9 18                LDAIM YMSG
306B 85 EA                STA   MSGPTL
306D A9 31                LDAIM YMSG    /
306F 20 D7 30             JSR   MSGWRT
3072 A5 F4                LDA   YREG    PRINT Y REGISTER
3074 20 3B 1E             JSR   PRTBYT
3077 A9 1C                LDAIM AMSG    MORE HEADING
3079 85 EA                STA   MSGPTL
307B A9 31                LDAIM AMSG    /
307D 20 D7 30             JSR   MSGWRT
3080 A5 F3                LDA   ACC
3082 20 3B 1E             JSR   PRTBYT  PRINT ACCUMULATOR
3085 A9 24                LDAIM PMSG
3087 85 EA                STA   MSGPTL
3089 A9 31                LDAIM PMSG    /
308B 20 D7 30             JSR   MSGWRT
308E A2 07                LDXIM $07
3090 A5 F1                LDA   PREG    SAVE PREG
3092 48                   PHA           LOOP TO PRINT THE

3093 26 F1        PLOOP   ROL   PREG    STATUS REGISTER
3095 B0 04                BCS   WUN     IN BINARY
3097 A9 30                LDAIM '0
3099 D0 02                BNE   OUTP
309B A9 31        WUN     LDAIM '1
309D 20 A0 1E     OUTP    JSR   OUTCH
30A0 CA                   DEX
30A1 10 F0                BPL   PLOOP
```

```
30A3 68                PLA
30A4 85 F1             STA    PREG    RESTORE PREG
30A6 A5 E9             LDA    SLOMO   SLOW MOTION IN EFFECT?
30A8 F0 1F             BEQ    NOSLO   NO
30AA 20 2F 1E          JSR    CRLF
30AD 20 2F 1E          JSR    CRLF
30B0 A6 E9             LDX    SLOMO   YES, USE AS COUNT
30B2 A9 F4     TLUPA   LDAIM  $F4     ABOUT A QUARTER SECOND
30B4 8D 07 17          STA    TIMST   START TIMER
30B7 AD 07 17  TLUPB   LDA    TIMOUT  WAIT FOR TIMEOUT
30BA F0 FB             BEQ    TLUPB
30BC CA                DEX            COUNT
30BD D0 F3             BNE    TLUPA
30BF 20 6A 1F          JSR    GETKEY  READ KEYBOARD
30C2 C9 15             CMPIM  $15     ANY KEY PRESSED?
30C4 D0 03             BNE    NOSLO   YES, QUIT
30C6 4C C8 1D          JMP    GOEXEC  NO, EXECUTE NEXT INSTRUCTION
30C9 20 2F 1E  NOSLO   JSR    CRLF
30CC 4C 4F 1C          JMP    START   RETURN TO KIM



               PSAVY : PRINT HEX SAVING THE Y REGISTER

30CF 84 EE     PSAVY   STY    SAVY
30D1 20 3B 1E          JSR    PRTBYT
30D4 A4 EE             LDY    SAVY
30D6 60                RTS

               MSGWRT : PRINT STRING AT MSGPTR TO FIRST NULL

30D7 85 EB     MSGWRT  STA    MSGPTH  HI POINTER BYTE IN A
30D9 A0 00     MSGLUP  LDYIM  $00
30DB B1 EA             LDAIY  MSGPTR  GET A BYTE
30DD F0 08             BEQ    MSGDUN  QUIT ON NULL
30DF 20 A0 1E          JSR    OUTCH   PRINT IT
30E2 E6 EA             INC    MSGPTL
30E4 D0 F3             BNE    MSGLUP  INCREMENT POINTER
30E6 E6 EB             INC    MSGPTH
30E8 D0 EF             BNE    MSGLUP
30EA 60        MSGDUN  RTS



                  MESSAGES

30EB 0D        VMSG      =    CR
30EC 0A                  =    LF
30ED 0A                  =    LF
30EE 0A                  =    LF
30EF 0A                  =    LF
30F0 0A                  =    LF
30F1 20                  =    '
30F2 56                  =    'V
30F3 49                  =    'I
30F4 5A                  =    'Z
30F5 41                  =    'A
30F6 2D                  =    '-
30F7 4B                  =    'K
30F8 49                  =    'I
30F9 4D                  =    'M
30FA 0D                  =    CR
30FB 0A                  =    LF
30FC 0A                  =    LF
30FD 20                  =    '
30FE 20                  =    '
30FF 50                  =    'P
3100 43                  =    'C
3101 20                  =    '
3102 20                  =    '
3103 20                  =    '
3104 44                  =    'D
3105 41                  =    'A
3106 54                  =    'T
3107 41                  =    'A
3108 0D                  =    CR
3109 0A                  =    LF
310A 20                  =    '
310B 00                  =    ZERO
310C 0D        SPMSG     =    CR
310D 0A                  =    LF
310E 53                  =    'S
310F 50                  =    'P
3110 3D                  =    '=
3111 00                  =    ZERO
3112 0D        XMSG      =    CR
3113 0A                  =    LF
3114 20                  =    '
3115 58                  =    'X
3116 3D                  =    '=
3117 00                  =    ZERO
3118 20        YMSG      =    '
3119 59                  =    'Y
311A 3D                  =    '=
311B 00                  =    ZERO
311C 0D        AMSG      =    CR
311D 0A                  =    LF
311E 20                  =    '
311F 20                  =    '
3120 20                  =    '
3121 41                  =    'A
3122 3D                  =    '=
3123 00                  =    ZERO
3124 0D        PMSG      =    CR
3125 0A                  =    LF
3126 20                  =    '
3127 20                  =    '
3128 20                  =    '
3129 20                  =    '
312A 20                  =    '
312B 50                  =    'P
312C 0D                  =    CR
312D 0A                  =    LF
312E 20                  =    '
312F 4E                  =    'N
3130 56                  =    'V
3131 20                  =    '
3132 42                  =    'B
3133 44                  =    'D
3134 49                  =    'I
3135 5A                  =    'Z
3136 43                  =    'C
3137 0D                  =    CR
3138 0A                  =    LF
3139 20                  =    '
313A 00                  =    ZERO
```

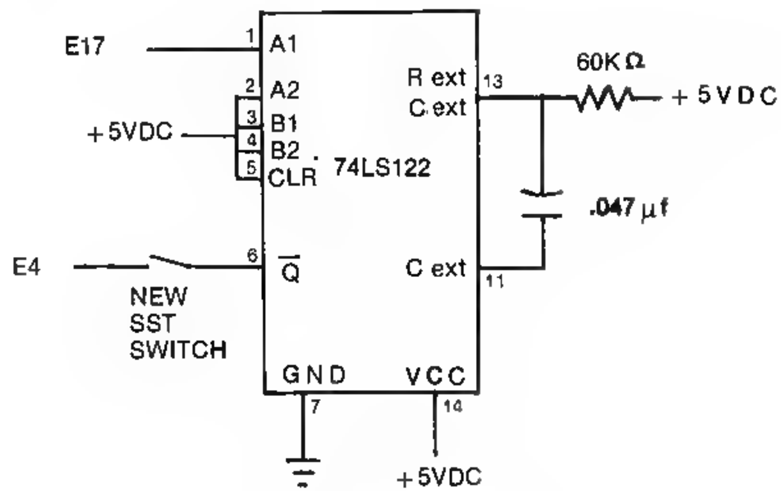
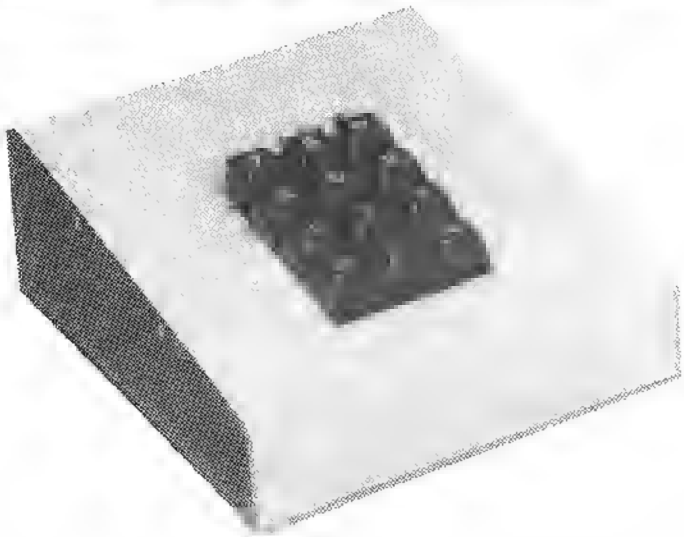



*Figure 1:* **VIZA-KIM Circuit**

# OSI



**BKM'S NUMERIC KEYPAD — $65**
**FOR OHIO SCIENTIFIC OR LEAR SIEGLER'S ADM-3A**

**FEATURES**

Digits 0-9, Decimal Point, and Return
**Terminal** Quality Keypad
Ready to use with C4's and C8's
Two Color Beige-Brown Case
Size 7"D x 5"W x 2½"H
Documentation

**OPTIONS**

Adapter cable kit for
10 minute INSTALLATION — $4.95
SPECIFY

**Superboard** **C1**
**C2-4P** **C2-8P**

WE PAY SHIPPING TO CONTINENTAL U.S.
TEXAS RESIDENTS ADD 5%
ORDER DIRECT OR ASK YOUR OSI DEALER
FOREIGN ORDERS ADD 30%

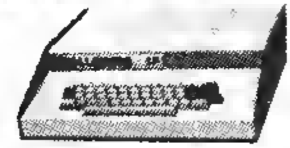
BKM MICRO SYSTEMS CORPORATION

VISA master charge

**3809 OLD COLLEGE ROAD**
**BRYAN, TEXAS 77801**

**713/846-8268**
**713/846-3817**

# OHIO SCIENTIFIC



# IN STOCK

| | |
|---|---|
| *SUPERBOARD II 8K Basic* | *299.00* |
| *CHALLENGER 1P 8K Ram* | *399.00* |
| *CHALLENGER 1P Minifloppy* | *1250.00* |
| *C4P Cassette Color 8K Ram* | *750.00* |
| *C4P Minifloppy 24K Color* | *1795.00* |
| *C8P Expandable 8K Ram* | *950.00* |
| *C8PDF Dual 8" Floppies 32K* | *2895.00* |
| *C2DEM 48K Dual 8" Floppies* | *2799.00* |
| *C3DEM 48K Triple Processor* | *3995.00* |
| *C3S1 Dual 8" Floppies 48K* | *4095.00* |
| *C3A 48" Cabinet C3* | *5995.00* |
| *DSK-5A 5 Ft. Desk for C2DEM* | *300.00* |
| *AC-3P B&W 12" TV/Vid. Mon.* | *129.00* |
| *AC-15P 12" Color Monitor* | *475.00* |
| *AC-9TP Centronics Tractor Ptr* | *1250.00* |
| *AC11P 300 Baud Orig. Ans. Mod.* | *199.00* |
| *AC-14 NEC Spinwriter 55cps* | *2795.00* |
| *AC-5A Okidata HS Line Ptr* | ** 2950.00* |
| *S-120 Soroc Serial Video Term.* | *995.00* |
| *CA-9 Centronics Parallel Interf.* | *175.00* |
| *CA-10-2 2 RS232 Serial Ports* | *175.00* |
| *CA-12 96 Line Parallel I/O* | *125.00* |
| *CA-14A Votrax Voice Synthesizer* | *399.00* |
| *4KP 4K Expansion Kit of 2114's* | *49.00* |
| *CM-2 4K Expsion Mem for D000* | *129.00* |
| *CM-3A 16K Low Power 2Mhz Bd.* | *399.00* |
| *CM-6 48K Dynamic Ram 1 Mhz.* | *549.00* |
| *CM-9 24K Static Ram 2114's* | *450.00* |
| *CM-10 8K Static Ram D&E Addr.* | *198.00* |
| *610-8K Supbd 8K Exp. & Disc Ctl* | *298.00* |
| *ATV Microverter to Modulate TV* | *35.00* |
| *505 B Upgrade Disk Ctlr CPU Bd* | *275.00* |
| *540B Color Video Board* | *200.00* |
| *510C Upgrade to C3 CPU's* | *600.00* |
| *470B Upgrade Floppy Controller* | *135.00* |
| *C730 Centronics 730 Parallel Ptr.* | *899.00* |

*At time of preparation all of the above items were in stock except those marked with an *.*

# FREE ATV MICROVERTER

With purchase of an 8K

CHALLENGER I P

Limited Quantity--First 25 Customers

## COMPUTERSHOP

| Boston | Union N.H. | Cambridge |
|---|---|---|
| 590 Comm. Ave. | Rte. 16B | 288 Norfolk St. |
| (across from B.U.) | 603-473-2323 | (near M.I.T.) |
| 247-0700 | | 661-2670 |

# Microbes & Updates

*Bill Watts of Provincetown, Mass phoned in the following changes to Henk Wevers' article "Shorthand Commands for Superboard II and Challenger C1P BASICs" (24:25):*

Page 26:

Line 028B Restore ↑H 68

Line 028F should be 67

0291 should be 65,

0292: 61

0295: 64

0298: 62

029A: 63

029E: 66

Page 27: Line 0236 should read A2 58, instead of A2 43.

With these changes, things should run smoothly.

*Bill Crouch from California writes:*

Line 63000 of the program XFILE.MAKER (23:11) was sent as "63000 REM XFILE.MAKER". The typesetter dropped the line number and used it as a title. The programs will not work unless there is a line 63000 in XFILE.MAKER so some of your readers might have problems with it.

Also, if you want to use REM KILLER on a program which has GOTO and GOSUB statements which refer to remark lines, you can change line 310 of REM KILLER to read:

```
310 PRINT ARRAY(Y);CHR$(58)
```

This will replace the REM statements with a colon. Although it doesn't save as much space as a complete removal of the REMs, the program will still work as before.

*From Robert and Jon Prall of Silver Spring, Maryland found a problem in "Apple II Speed Typing Test with Input Time Clock" in the December issue of 1979.*

On page 19:69 line 8406 reads in the published version, subtracting 159 from ASCII numbers assigned to the individual characters does not correspond to the position of characters the A$.

The inclusion of the quotation mark at position three in the string is logical, but impossible because it causes a "Syntax Error" message, and a blank space should be substituted for it.

The corrected line should read:

```
8406 A$ = " ! #$%&'()* + ,-.
/0123456789:;< = >
?@ABCDEFGHIJKLMNOPQRSTUVWXYZ"
```

The position of the spaces in the string is essential; the signs for greater than and less than must be included, as must the exclamation point. The author's inclusion of the slash, the small 'm' amd a space at the end of the string appear to be additional errors.

With the corrections noted, the program runs very well.

*Rev. James Strasma sends this update to his article entitled "Lower Case Lister" (25:11):*

A revised printer ROM is now available for CBM printers without charge. It improves lower-case listings. However, the 20 characters that failed to print correctly in lower-case mode before stil fail. "Lower Case Lister" is compatible with the new '04' printer ROM., and corrects all characters.

# Challenger II Communications

**Everything you naad to turn your OSI with a 502 CPU board into a 'standerd' communications terminal: hardwara changas end tha softwara to run it.**

**Peter Koski**

As a college student, time becomes extremely valuable. A very poor use of this time is sitting waiting for a computer terminal. Corollary to Murphy's Law — there are never enough terminals; and who uses cards in this day and age?

Looking logically at the situation, there was only one answer, and my OSI Challenger II was it. Generously enough, Ohio Scientific has provided their 502 CPU board with all the foils needed for serial TTL/RS-232 input/output.

My answer was found. While others are sitting at terminals till the wee hours of the morning, I can be happily talking to Myron (our resident IBM) from the comfort of my room. Stereo in the background, fridge to the right ... what a life!

Of course this also opens up a whole horizon of dial-up bulletin board services as well as time-share systems. Options no computerist should live without.

### Hardware

Before any software can be written, we'd better have some hardware to play with. Conveniently enough, the cassette port runs at 300 baud. No problem here. What about the -9 volts required by RS-232? Again we're allright: most modems only require a swing to zero level. Great!

First, let's start with the output side of the problem. Locate, using OSI's 502 schematic package, the positions of U31, R55, R56, R57 and Q2. Some boards may or may not have U31 on them already. If not, install U31 using an I.C socket. The values for the parts may be summarized:

| | | |
|---|---|---|
| U31 | 7404 | (hex inverter) |
| R55, 57 | 10KΩ | (¼ watt) |
| R56 | 470Ω | (¼ watt) |
| Q2 | 2N5226 | |

Carefully solder these to the board, confirming the positions. Check for any solder bridges which may crop up.

Input becomes only a bit more complicated. In order to maintain cassette capability, a switch must be inserted in the ACIA input line (the cassette input ciruit loads down the line). Any SPDT switch which fits on the rear apron will suffice (Radio Shack's paddle switches fit the 3/4 inch holes perfectly). Install the switch and we'll worry about wiring it later.

Again referring to the 502 layout sheet, locate U20, R61, R62, D3, and Q4. As with U31, U20 may or may not exist already. If not, be sure to use a socket when installing it. Once their positions are located, the following parts may be installed:

| | | |
|---|---|---|
| U21 | 7404 | (hex inverter) |
| R61 | 10KΩ | (¼ watt) |
| R62 | 4.7KΩ | (¼ watt) |
| D3 | 1N914 | |
| Q4 | 2N5225 | |

Be certain the board looks right before continuing on.

Going to pin 2 of the ACIA (U3) is the RX DATA foil. Cut this foil at some convenient point and solder the center terminal lead of the switch to the ACIA side of the cut. Solder one of the other leads to the other side of the break. In this switch position, cassette operation is as normal. Back to the newly installed U20. Locate the foil from pin 2 and cut it. To the U20 side of this foil, solder the remaining lead of the select switch. In this switch position, RS-232 input will be routed to the ACIA. A good thought would be to install a 3-pin in-line connector somewhere between the board and the select switch.

A standard RS-232 connector may also be added to the rear apron. The RX DATA is now at pin 1 of connector J3 and TX DATA at pin 7 of J3. All the even pins of J3 are ground. (-9 volts is bussed on the backplane, just add your supply if needed).

Unless you feel confident in your soldering abilities, you may want to let a trustworthy friend do the work for you. It only takes half an hour or so, but errors could be disasterous — and it's your own fault.

What you are now left with is an RS-232 port which resides at FC00 (same as cassette port). The input is selectable: cassette or 300 baud RS-232. Output is always there, allowing for convenient printer listings of programs being SAVEd to

tape. The uses and tricks that can be inplemented are too numerous to list; you'll find them yourself.

As for the modem, the Novation Cat is probably the top of the line if you can afford it. I have used it with excellent results over phone lines which would have made speech recognition rough, and I have not lost a bit. Plus it offers answer in addition to originate mode.

#### Software

Two options are now possible, and I've tried both. OSI's BASIC is fast enough to service the port via PEEKs and POKEs. However the draw-back is that it is very difficult to output BASIC control symbols (comma or colon). A BASIC routine is the easiest route if you wish to set up a system for down-loading locally-editted files. This is a very handy routine which works well. See the two BASIC programs below.

On the other hand, the following assembler routine turns your brilliant computer into an ignorant terminal. Running with this system, the Challenger II behaves like a standard ASCII terminal, except the obscure CTRL functions will appear as OSI graphics.

The package includes a protected field at the top of the screen to provide a 'touch of class' without taking too much screen space.

As written, the routine is loaded into 2000 hex. However, it could be relocated fairly easily. The only monitor routine called is the keyboard input routine, whose entry point in the 65V MONITOR is FEED hex (should be the same for all systems). The program continually polls both the port and the keyboard, then displays or output (as the case may be) whichever is requesting service at the time. Auto-line feed is provided only on out-put (as the case may be) whichever is requesting service at the time. Auto-line feed is provided only on out-put carriage return. Most dial-ups will provide line-feed with carriage return.

As an added note of interest, the RS-232 outputs from both the Challenger II and modem are able to handle two loads. This means that a printer could be used on one line (normally input) to provide hard-copy as desired. Certainly no computer system should be without RS-232 communications capabilities.

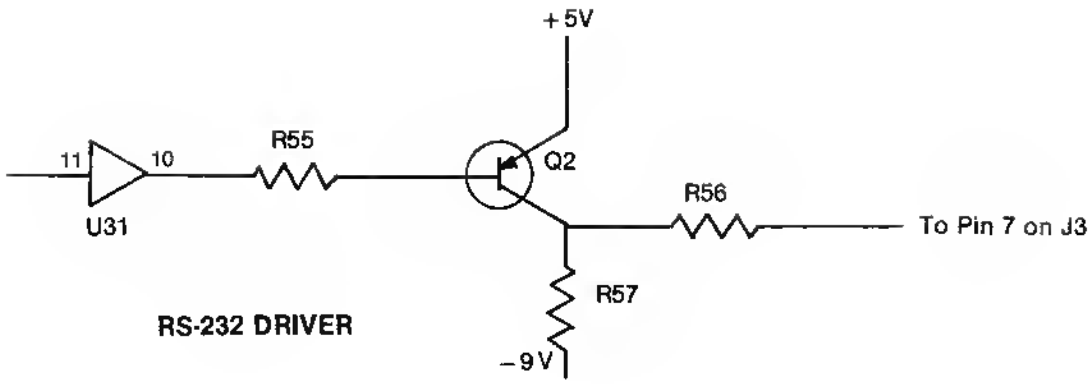
My system has behaved flawlessly through "mega-hours" of hard use. Good luck, and don't make Ma Bell too rich with your calls!

μ

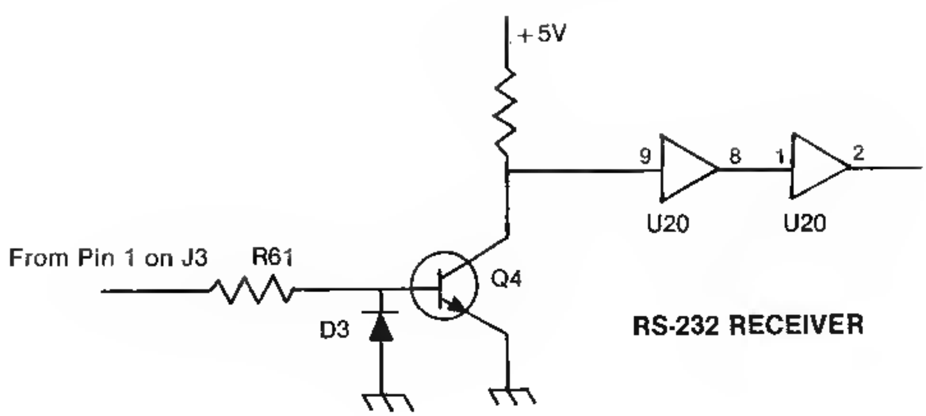
*Peter is a sophomore at Rensselaer Polytecnic Institute majoring in Biomedical Engineering — Electronics option. His minor is in Computer Systems. He has an Ohio Scientific Challenger C2-4P which he uses for both academic and hobby purposes. Pete started his programming in BASIC and recently added assembler capability to his machine's repetoire.*

```
1000 REM —        TERMINAL OPERATING SYSTEM
1010 REM
1020 REM                  VERSION  3.2
1022 REM
1025 REM — PETER KOSKI                   12/79
1030 REM
1035 REM — STRIKING EITHER 'SHIFT' KEY ENTERS
1037 REM    TRANSMIT MODE  ( ? PROMPT)
1038 REM
1040 REM — WHEN ENTERING A COMMA OR COLON, THE
1050 REM    'CTRL' MUST ALSO BE DEPRESSED
1060 REM
1090 REM — UNDERSCORE IS CONTROL HYPHEN
1100 REM
1110 REM — OR OPERATOR IS CONTROL I
1120 REM
1130 REM -- NOT OPERATOR IS #
1140 REM
2000 POKE 530,1: POKE 57088,1: POKE 64512,1
2010 IF (PEEK(64512)AND1)THEN PRINT CHR$(PEEK(64513));
2020 IF (PEEK(57088)=1) THEN 2010
2030 INPUT TX$
2040 FOR TX=1 TO LEN(TX$)
2050 FOR DLA=1 TO 15: NEXT DLA
2060 IF ASC(MID$(TX$,TX,1))=122 THEN POKE 64513,58: GOTO 2100
2065 IF ASC(MID$(TX$,TX,1))=108 THEN POKE 64513,44: GOTO 2100
2070 IF ASC(MID$(TX$,TX,1))=109 THEN POKE 64513,95: GOTO 2100
2080 IF ASC(MID$(TX$,TX,1))=113 THEN POKE 64513,124:GOTO 2100
2090 IF ASC(MID$(TX$,TX,1))= 35 THEN POKE 64513,126: GOTO 2100
2095 POKE 64513,ASC(MID$(TX$,TX,1))
2100 NEXT TX
2150 FOR DLA=1 TO 15: NEXT DLA: POKE 64513,13
2160 FOR DLA=1 TO 15: NEXT DLA: POKE 64513,10
2170 GOTO 2010
OK
```

```
1000 REM          TERMINAL OPERATING SYSTEM
1001 REM
1002 REM               VERSION  3.3
1003 REM
1004 REM --  PETER KOSKI                   11/79
1005 REM
1006 REM --  LOCAL FILE EDITOR / TERMINAL SYSTEM PACKAGE
1007 REM
1009 DIM LINE$(60), TEMP$(64)
1010 FOR CLS=1 TO 20: PRINT: NEXT CLS
1020 PRINT"        >>>>  TOS  VERSION  3.3   <<<<"
1030 PRINT:PRINT:PRINT "  ---  LOAD  (LOAD LOCAL FILE)"
1040 PRINT:PRINT "  ---  EDIT  (EDIT LOCAL FILE)"
1050 PRINT:PRINT "  ---  TMOD  (ENHANCED TERMINAL MODE)"
1060 PRINT:PRINT:PRINT:INPUT MOD$
1070 IF LEFT$(MOD$,4)="LOAD" THEN ID=1
1080 IF LEFT$(MOD$,4)="EDIT" THEN ID=2
1090 IF LEFT$(MOD$,4)="TMOD" THEN ID=3
1100 ON ID GOSUB 2000, 3000, 4000
1110 GOTO 1010
2000 REM -- LOAD LOCAL FILE
2010 FOR CLS=1 TO 14:PRINT:NEXT CLS
2015 FOR CM=1 TO 60: LINE$(CM)=CHR$(32): NEXT CM
2020 PRINT "     >>>  LOCAL  FILE  LOADER   <<<"
2025 PRINT:PRINT
2030 PRINT:PRINT "  ---  OR operator is CTRL-1"
2035 PRINT:PRINT "  ---  NOT operator is #"
2040 PRINT:PRINT "  ---  UNDERSCORE is CTRL HYPHEN"
2050 PRINT:PRINT "  ---  CTRL must be depressed when"
2060 PRINT "       entering a COMMA or COLON"
2070 PRINT:PRINT "  ---  $ENDFILE marks end-of-file"
2080 PRINT:PRINT: LN=1
2090 INPUT LINE$(LN)
2100 IF LEFT$(LINE$(LN),8)="$ENDFILE" THEN RETURN
2110 LN=LN+1: GOTO 2090
3000 REM -- EDIT LOCAL FILE
3010 FOR CLS=1 TO 14: PRINT: NEXT CLS
3060 PRINT "        >>>  LOCAL FILE EDITOR   <<<"
3070 PRINT:PRINT
3080 PRINT:PRINT "  ---  INSERT , LINE NUMBER PRECEEDING INSERT"
3090 PRINT "         LOCATION DESIRED"
3100 PRINT:PRINT "  ---  DELETE , LINE NUMBER TO BE DELETED"
3105 PRINT:PRINT "  ---  LIST
3110 PRINT:PRINT "  ---  DONE"
3120 PRINT:PRINT:INPUT "OPTION";OPTN$
3140 IF LEFT$(OPTN$,4)="LIST" THEN ID=3: GOTO 3180
3150 IF LEFT$(OPTN$,4)="DONE" THEN RETURN
3160 INPUT "  LINE";LINE
3170 IF LEFT$(OPTN$,6)="INSERT" THEN ID=1
3175 IF LEFT$(OPTN$,6)="DELETE" THEN ID=2
3180 ON ID GOSUB 3200, 3260, 3310
3190 GOTO 3120
3200 FOR B= (LN+1) TO (LINE+1) STEP -1
3210 LINE$(B)=LINE$(B-1)
3220 NEXT B
3230 PRINT:INPUT INSERT$
3240 LINE$(LINE+1)=INSERT$
3250 LN=LN+1: RETURN
3260 FOR C=LINE TO LN-1
3270 LINE$(C)=LINE$(C+1)
3280 NEXT C
3290 LINE$(LN)=CHR$(32)
3300 LN=LN-1: RETURN
3310 PRINT:PRINT: FOR D=1 TO LN
3320 PRINT D, LINE$(D)
3330 NEXT D: RETURN
4000 REM -- ENHANCED TERMINAL OPERATING SYSTEM
4010 FOR CLS=1 TO 10: PRINT: NEXT CLS
4020 PRINT "    >>>  ENHANCED TERMINAL OPERATING SYSTEM   <<<"
4030 PRINT:PRINT
4032 PRINT:PRINT "  ---  Striking either SHIFT key enters"
4035 PRINT"       TRANSMIT mode ( ? prompt )"
4040 PRINT:PRINT "  ---  OR operator is CTRL-1"
4050 PRINT:PRINT "  ---  NOT operator is #"
4055 PRINT:PRINT "  ---  UNDERSCORE is CTRL HYPHEN"
4060 PRINT:PRINT "  ---  CTRL must be depressed when"
4065 PRINT"       entering a COMMA or COLON"
4075 PRINT:PRINT "  ---  DUMP (DUMPS LOCAL FILE)"
4076 PRINT:PRINT "  ---  DONE"
4078 PRINT:PRINT
4080 POKE 530,1: POKE 57088,1: POKE 64512,1
4082 IF (PEEK(64512)AND1) THEN PRINT CHR$(PEEK(64513));
4083 IF (PEEK(57088)=1) THEN 4082
4084 INPUT TX$
4085 IF LEFT$(TX$,4)="DUMP" THEN 5000
4087 IF LEFT$(TX$,4)="DONE" THEN RETURN
4100 FOR TX=1 TO LEN(TX$)
4110 FOR DLA=1 TO 15: NEXT DLA
```




```
4120 IF ASC(MID$(TX$,TX,1))=122 THEN POKE 64513,58: GOTO 4200
4130 IF ASC(MID$(TX$,TX,1))=108 THEN POKE 64513,44: GOTO 4200
4140 IF ASC(MID$(TX$,TX,1))=109 THEN POKE 64513,95: GOTO 4200
4150 IF ASC(MID$(TX$,TX,1))=113 THEN POKE 64513,124:GOTO 4200
4160 IF ASC(MID$(TX$,TX,1))=35 THEN POKE 64513,126:GOTO 4200
4170 POKE 64513,ASC(MID$(TX$,TX,1))
4200 NEXT TX
4210 FOR DLA=1 TO 15: NEXT DLA: POKE 64513,13
4220 FOR DLA=1 TO 15: NEXT DLA: POKE 64513,10
4230 GOTO 4002
5000 REM -- LOCAL FILE DUMP ROUTINE
5010 FOR CLS=1 TO 28: PRINT: NEXT CLS
5020 PRINT "      >>>   LOCAL FILE DUMP ROUTINE   <<<"
5030 PRINT:PRINT:PRINT:PRINT
5050 FOR G=1 TO LN
5060 FOR H=1 TO LEN(LINE$(G))
5070 TEMP$(H)=MID$(LINE$(G),H,1)
5080 IF TEMP$(H)="l" THEN TEMP$(H)=","
5090 IF TEMP$(H)="{" THEN TEMP$(H)=";"
5100 IF TEMP$(H)="z" THEN TEMP$(H)=":"
5110 IF TEMP$(H)="a" THEN TEMP$(H)=CHR$(95)
5120 IF TEMP$(H)="q" THEN TEMP$(H)=CHR$(124)
5125 IF TEMP$(H)="#" THEN TEMP$(H)=CHR$(126)
5130 NEXT H
5140 LT=LEN(LINE$(G)): LINE$(G)=" "
5150 FOR I=1 TO LT: LINE$(G)=LINE$(G)+TEMP$(I): NEXT I
5155 FOR WT=1 TO 1200: NEXT WT
5160 POKE 517,255: PRINT RIGHT$(LINE$(G),LT): POKE 517,0
5180 NEXT G
5190 GOTO 4000
```



```
0010:                     CHALLENGER II
0020:                     TERMINAL OPERATING SYSTEM
0030:                     BY PETER KOSKI
0040:
0050: 2000                TOS     ORG   $2000
0060:
0070: 2000 A9 20                  LDAIM $20
0080: 2002 A0 08                  LDYIM $08
0090: 2004 A2 00                  LDXIM $00
0100:
0110: 2006 9D 00 D0       LOOP    STAX  $DC00
0120: 2009 E8                     INX
0130: 200A D0 FA                  BNE   LOOP
0140: 200C EE 08 20               INC   $2008
0150: 200F 88                     DEY
0160: 2010 D0 F4                  BNE   LOOP
0170: 2012 A9 D0                  LDAIM $D0
0180: 2014 8D 08 20               STA   $2008
0190: 2017 4C 41 20               JMP   $2041
0200: 201A 43                     =     'C
0210: 201B 48                     =     'H
0220: 201C 41                     =     'A
0230: 201D 4C                     =     'L
0240: 201E 4C                     =     'L
0250: 201F 45                     =     'E
0260: 2020 4E                     =     'N
0270: 2021 47                     =     'G
0280: 2022 45                     =     'E
0290: 2023 52                     =     'R
0300: 2024 20                     =     '
0310: 2025 20                     =     '
0320: 2026 49                     =     'I
0330: 2027 49                     =     'I
0340: 2028 54                     =     'T
0350: 2029 45                     =     'E
0360: 202A 52                     =     'R
0370: 202B 4D                     =     'M
0380: 202C 49                     =     'I
0390: 202D 4E                     =     'N
0400: 202E 41                     =     'A
0410: 202F 4C                     =     'L
0420: 2030 20                     =     '
0430: 2031 4F                     =     'O
0440: 2032 50                     =     'P
0450: 2033 45                     =     'E
0460: 2034 52                     =     'R
0470: 2035 41                     =     'A
0480: 2036 54                     =     'T
0490: 2037 49                     =     'I
0500: 2038 4E                     =     'N
0510: 2039 47                     =     'G
0520: 203A 20                     =     '
0530: 203B 53                     =     'S
0540: 203C 59                     =     'Y
0550: 203D 53                     =     'S
0560: 203E 54                     =     'T
0570: 203F 45                     =     'E
0580: 2040 4D                     =     'M
0590: 2041 A2 0E                  LDXIM $0E
0600: 2043 BD 19 20       LOOPB   LDAX  $2019
0610: 2046 9D EC D0               STAX  $D0EC
0620: 2049 CA                     DEX
0630: 204A D0 F7                  BNE   LOOPB
0640: 204C A2 19                  LDXIM $19
0650: 204E BD 27 20       LOOPC   LDAX  $2027
0660: 2051 9D 61 D1               STAX  $D161
0670: 2054 CA                     DEX
0680: 2055 D0 F7                  BNE   LOOPC
0690: 2057 A2 40                  LDXIM $40
0700: 2059 A9 94                  LDAIM $94
0710: 205B 9D BF D1       LOOPD   STAX  $D1BF
0720: 205E CA                     DEX
0730: 205F D0 FA                  BNE   LOOPD
0740: 2061 AD 00 FC       LOOPE   LDA   $FC00
0750: 2064 4A                     LSRA
0760: 2065 B0 1E                  BCS   LOCA
0770: 2067 EA                     NOP
0780: 2068 A9 02                  LDAIM $02
0790: 206A 8D 00 DF               STA   $DF00
0800: 206D AE 00 DF               LDX   $DF00
0810: 2070 D0 20                  BNE   LOCB
0820: 2072 0A                     ASLA
0830: 2073 F0 EC                  BEQ   LOOPE
0840: 2075 4C 6A 20               JMP   $206A
0850: 2078 A0 B9                  LDYIM $B9
0860: 207A A2 00                  LDXIM $00
0870: 207C C8             LOOPF   INY
0880: 207D F0 E2                  BEQ   LOOPE
0890: 207F E8                     INX
0900: 2080 F0 FA                  BEQ   LOOPF
0910: 2082 4C 7F 20               JMP   $207F
0920: 2085 AD 01 FC       LOCA    LDA   $FC01
0930: 2088 29 7F                  ANDIM $7F
0940: 208A F0 D5                  BEQ   LOOPE
0950: 208C 20 BC 20               JSR   $20BC
0960: 208F 4C 61 20               JMP   $2061
0970: 2092 20 ED FE       LOCB    JSR   $FEED
0980: 2095 C9 0D                  CMPIM $0D
0990: 2097 F0 09                  BEQ   LOCD
1000: 2099 8D 01 FC               STA   $FC01
1010: 209C 20 BC 20               JSR   $20BC
1020: 209F 4C 78 20               JMP   $2078
1030: 20A2 8D 01 FC       LOCD    STA   $FC01
1040: 20A5 20 BC 20               JSR   $20BC
1050: 20A8 A0 0A                  LDYIM $0A
1060: 20AA AD 00 FC       LOCE    LDA   $FC00
```

```
1070: 20AD 4A             LSRA
1080: 20AE 4A             LSRA
1090: 20AF 90 F9          BCC   LOCE
1100: 20B1 8C 01 FC       STY   $FC01
1110: 20B4 A9 0A          LDAIM $0A
1120: 20B6 20 BC 20       JSR   $20BC
1130: 20B9 4C 78 20       JMP   $2078
1140: 20BC C9 0D          CMPIM $0D
1150: 20BE F0 19          BEQ   LOCF
1160: 20C0 C9 0A          CMPIM $0A
1170: 20C2 F0 28          BEQ   LOCG
1180: 20C4 AE D8 20       LDX   $20D8
1190: 20C7 9D 00 D7       STAX  $D700
1200: 20CA A9 20          LDAIM $20
1210: 20CC 9D 40 D7       STAX  $D740
1220: 20CF EE D8 20       INC   $20D8
1230: 20D2 A9 87          LDAIM $87
1240: 20D4 9D 41 D7       STAX  $D741
1250: 20D7 60             RTS
1260: 20D8 00             BRK
1270: 20D9 A9 20    LOCF  LDAIM $20
1280: 20DB AE D8 20       LDX   $20D8
1290: 20DE 9D 40 D7       STAX  $D740
1300: 20E1 A9 87          LDAIM $87
1310: 20E3 8D 40 D7       STA   $D740
1320: 20E6 A9 00          LDAIM $00
1330: 20E8 8D D8 20       STA   $20D8
1340: 20EB 60             RTS
1350: 20EC AE 40 D2 LOCG  LDX   $D240
1360: 20EF 8E 00 D2       STX   $D200
1370: 20F2 18             CLC
1380: 20F3 AD ED 20       LDA   $20ED
1390: 20F6 69 01          ADCIM $01
1400: 20F8 8D ED 20       STA   $20ED
1410: 20FB AD EE 20       LDA   $20EE
1420: 20FE 69 00          ADCIM $00
1430: 2100 8D EE 20       STA   $20EE
1440: 2103 18             CLC
1450: 2104 AD F0 20       LDA   $20F0
1460: 2107 69 01          ADCIM $01
1470: 2109 8D F0 20       STA   $20F0
1480: 210C AD F1 20       LDA   $20F1
1490: 210F 69 00          ADCIM $00
1500: 2111 8D F1 20       STA   $20F1
1510: 2114 AD EE 20       LDA   $20EE
1520: 2117 C9 D7          CMPIM $D7
1530: 2119 90 D1          BCC   LOCG
1540: 211B AD ED 20       LDA   $20ED
1550: 211E C9 3F          CMPIM $3E
1560: 2120 EA             NOP
1570: 2121 90 C9          BCC   LOCG
1580: 2123 A2 00          LDXIM $00
1590: 2125 A9 20          LDAIM $20
1600: 2127 9D 00 D7 LOOPZ STAX  $D700
1610: 212A E8             INX
1620: 212B E0 40          CPXIM $40
1630: 212D 90 F8          BCC   LOOPZ
1640: 212F A9 40          LDAIM $40
1650: 2131 8D ED 20       STA   $20ED
1660: 2134 A9 00          LDAIM $00
1670: 2136 8D F0 20       STA   $20F0
1680: 2139 A9 D2          LDAIM $D2
1690: 213B 8D EE 20       STA   $20EE
1700: 213E 8D F1 20       STA   $20F1
1710: 2141 60             RTS
ID=
```

## Classified Ads

KIM Basic users: upgrade to full-featrd Basic with renumb, append, improved editor, file system supporting PET-like file commands & more. Incl casstt, manual, sample progs, compl source list. Many practical applica to KIM Basic. Send $43. for packg or SASE for 3 pg compl descr.

Sean McKenna
64 Fairview Av.
Piedmont, CA 94610

PET MACHINE LANGUAGE GUIDE: Comprehensive manual to aid mach. lang. progrmmr. More than 30 routns fully detailed: reader can put to immed. use. For New or Old ROMS. $6.95 plus .75 p&h. VISA/Mastercharge accptd.

Abacus Software
P.O. Box 7211
Grand Rapids, MI 49510

Apple II Sweet 16 Assembler SW16 in mach lang & intg basic progs. Understand & use powerful 16 bit processor. Incl in disk & casstt based text ed is SW16 assemb. & users' man. Manual cntns treatment of SW16 prcessng, OP code descr. & more. Send $15. ($5. for man applic to assemb)

Scientific Software
P.O. Box 156
Stowe, PA 19464

British Apple Owners/Dealers: Write now to MGA for extensive list of specialised software and hardware for your Apple or 2020. We promise you'll be surprised!!

Michael Gurr Associates
140 High Street
Tenterden, Kent
TN30 6HT, England

OSI C1P Superboard II owners, you need the 96 page tutorial manual 'Getting Started with Your C1P'. Fundamentals of BASIC, cassette usage, subroutines, logic & control are described in step-by-step manner.$5.95 + $1 p&h form:

TIS
Box 921—M
Los Alamos, NM 87544

The Relationship Life Dynamic (Apple II Plus, 48K, $15.95:disk) A unique program for those who desire to experience transformation in their relationships w/o paying the high costs of commercial 'trips'. Incl games & HIRES graphics animation. Order from:

Avant-Garde Creations
P.O. Box 30161, Dept. MC
Eugene, OR 97403

AIM-65 system for sale Includes Power A Plus power supply. Used less than 5 hours. All documentation included: $375. Call (617) 924-0972, or write:

Edwin M. Kellogg
100 Robbins Road
Watertown, MA 02172

OSI Machine Code Renumber Program: selective renumb all, part of Basic prog. Fast, easy, no more modifica req. Avail self-load. checksum tape, or disk. Disk sys incl. utility routines. Tape:$6.95, Disk:$9.95. Specify sys & amt of memory.

L&J Personal Computing
2606 Grand Avenue
Grand Junction, CO 81501

AppleII Shape Table Editor makes shape constr & edit easy. 11 edit commands allow creation, modifca of shapes. Save & retr tables to use w/ DRWA, XDRAW & SHLOAD commnds of AP II Basic. Runs under firmwr Aplsft fl pt Basic w/32K RAM. Cassette & manual:$29.95, or $2 (refund w/ purchase) for maunal only.

Small Systems Software
P.O. Box 40737
Washington, DC 20016

OSI SOFTWARE: tapes for challenger C1P, superboard. startrek; starfighter & lunar lander; egyptian ruler; home budget. Each $7.50. catalog SASE.

JDS Software
2334 Antigua Ct.
Reston, VA 22091

AIM 65 Newsletter—hardware and Software, Utilities as well as Applications. Keep up to date in the AIM 65 world. Target is published bimonthly. Six issues $6.00 in US & CAN ($12. elsewhere)

Target
c/o Donald Clem
RR Number 2
Spencerville, OH 45887

TAPE MONITOR CONTROL C1P and Superboard usres. Turns recorder on and off with LOAD and SAVE. Adds on to, rather than modifying board. Parts cost about $10. Complete plans:$4.00

Bruce Miller
13325 W. Crawford Drive
New Berlin WI 53151

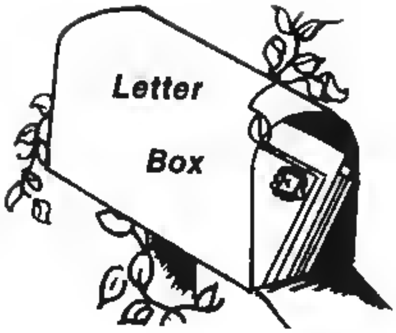
TRACER ($11.95) and other programs available. Phone or write:

Quality Software
3194 Ospika Pl
Prince George, BC
Canada V2N-2TS
(604) 563-9839

APPLE BARGAINS! EASY AS 1,2,3!
1)D.C HAYES MICROMODEM:$320
2)Z-80 SOFTCARD by MICROSOFT: Run CP/M on APPLE : $320
3)APPLE DATA-GRAPH

Screen, only : $25,
or FREE with hardware order!
C.O.D. ok

Connecticut Info. Systems
218 Huntington Road
Bridgeport, CT 06608
(203) 579-0472
CL0924 on SOURCE




*The following letters are in response to the editorial that appeared in the March issue of Micro. The editorial encouraged readers to write to us about what they'd like to see in a 6516. Here are two of those responses.*

Dear Bob,

I just read the March issue, and I am responding to your editorial on the "want list" for a 6516. Here's my list, with the most-wanted features first:

1. Let all op-codes use all possible addressing modes, so I won't need a wall chart to tell me if this op-code is allowed to use this addressing mode. Haven't you ever written a neat piece of code using, for example, ASLIY (Indirect Indexed), only to find that ASLIY isn't alowed? I may never use INCAY (Absolute Indexed by Y), but I sure would like to know that it's there if I ever want it. In my opinion, this is the best feature of the new 6809: there are no "holes" in the op-code-versus-address-mode matrix.

2. Change "Zero Page" to "Fast Page", and add the instruction SFP XX (set Fast Page). With the 6502, page zero is prime real estate. With this change, I can turn any page into prime real estate.

3. BRA (Branch Always). This only saves one byte per use (over CLC, BCC), but those bytes do add up.

4. BAS (Branch Always to Subroutine). In other words, a relative JSR. This would allow relocatable code without the hassle of subroutine-address look-up tables and zero-page trickery.

5. INA, DEA. Increment and decrement accumulator.

6. PHX, PLX, PHY, PLY. Push and pull X and Y.

7. EAX, EAY, EXY. Exchange A&X, A&Y, X&Y.

8. SSP XX (Set Stack Page). This would make the use of multipe stacks a lot easier.

9. DEL XX (Delay XX Cycles). Better yet, make it DEL XX XX. This would be neater than wait loops, or strings of NOPs and such when equalizing branches. Even better, DEL NN XX..., where NN designates number of following bytes that define delay time.

10. With all of the above, who needs 16 bytes?

Mel Evans
Ann Arbor, MI

Dear Dr.Tripp,

I am responding to your question concerning a revised or improved 6502. My first request would be to fill in all those presently used OP codes. I really need more indirect addressing modes like

LDA ($1234)
STA ($1234)
*[absolute indirect without index]*

I would also like an increment (and decrement) instruction which automatically adds the carry into the next byte. I guess this is a 16 byte instruction.

Of course PHX and PLX would also be helpful to save a few bytes. A new chip would have to be hardware compatible with my present system or I would have no real interest in it.

I heard that serveral years ago MOS Technology had some experimental improved 6502's However, this program ended when they were brought out by Commodore.

Dr.Morris
Midland, MI

*I had really expected to receive more suggestions on improvements for the 6502. Does the limited response indicate that you are all* ***totally*** *satisified with the 6502 as it is? That* ***would*** *suprize me! Even if you only have one small but significant idea, let us know about it. It could make a difference to the future development of the 6502.*

μ

# AIM 65 File Operations

**AIM BASIC doas not heve any flla eccess statemants. A discussion of this problem end programs to solva It ara prasentad. Thase programs will graatly anhanca the AIM BASIC, and provida soma insight into tha workings of tha AIM.**

**Christopher J. Flynn**

### Introduction

By now, most readers of MICRO are familiar with the physical characteristics of the Rockwell AIM 65 microcomputer. The AIM 65 is a computer which comes complete with keyboard, display, and a printer. A few additional ICs will add Microsott BASIC, a two-pass assembler, and an extra 3K ot RAM. All of this can be housed in an attractive case. The result is a truly personal computer. It can be easily moved around the home or oftice to where the user is. There is no concern about detached video monitors, expansion interfaces, cables, and the like. The AIM is indeed a very versatile computing engine.

This attractiveness ot the AIM 65 hardware was the factor that ultimately prompted my wife and me to purchase one. We quickly learned how to operate it. It comes with a one inch thick users manual! Rockwell deserves a lot of credit for not only paying attention to documentation, but also tor doing such a good job with It.

Upon contemplating our first home applications, we discovered that not much had been written about the application sottware capabilities of the AIM. We were happily creating data bases with the very nitty built-in text editor. Our intention was to next use BASIC to perform the desired calculations on the data. This is where we ran into a problem. AIM's BASIC has no file access statements! None of the provided documentation or any other 6502 sources could provide an answer to this dilemma. Did that mean that all that data which we had entered was useless? We will show that the answer to this question is a resounding NO!

We have developed a simple machine language subroutine. This subroutine will allow a BASIC program to read any AIM 65 text file. This includes data entered from the text editor as well as BASIC source program tapes themselves. The subroutine is easy to use. It does some error checking to prevent simple mistakes from ruining your day. It will also tell BASIC when the end of a file has been reached. As a bonus, the subroutine is completely position-independent and ROMable.

### Definitions

Before describing our software, we will define a few commonly used terms in AIM 65 context. This will benefit individuals who are just learning to use their AIM's and also MICRO readers who may not be aware ot the AIM's capabilities.

**File:** A file is a collection of data. AIM 65 files may reside on external media such as audio tape or paper tape. AIM 65 audio tape files may, in turn, be in AIM or KIM format. We will be concerned only with AIM 65 format audio tape tiles.

Each file is given a tile name. The file name may be from one to tive characters long.

There are two types of AIM 65 audio tape tiles. One type contains object code data. The other type contains text (or ASCII) data. The subroutine we are presenting will handle only text files.

The AIM 65 has a dual cassette interface. A file may be read (or written) from either drive number 1 or drive number 2. Incidentally, we have found this feature to be very handy.

**Block:** A block is the unit of Information transterred to and from memory and the audio cassette recorder.

All AIM format tape files are blocked. The tormat of text file blocks is described in the Users Manual. Suttice it to say, each block in any given tile will contain the same number of bytes. (The exact block length is a function ot the number of leading SYN characters.) Each block, though, will always contain 79 bytes of text data. It necessary, the last block will be padded with zeroes.

**Line or Record:** A line or record is the unit of information transterred to and from a program and the AIM monitor.

In a text file the lines will naturally contain ASCII data. The maximum line length can vary. The text editor imposes a 60 character limit on lines, while BASIC limits lines to 72 characters. The end ot a line, in either case, is marked with a carriage return.

Now here is where it gets tricky. Each block will always contain 79 data bytes. Since the lines can vary in length, a line may be either wholly contained within a block or it may span a block. The machine decides if a line will fit in a block. If not, the line is split in two. This may sound imposing, but don't worry about it. We'll show how this situation is handled later.

**End of File:** The occurrence of two successive carriage returns on a text file denotes that there are no more lines of data on the file. Upon detection of end of file, we want the BASIC program to stop and not to attempt any more read operations.

**Machine Language Subroutine:**
"Although Basic is a high level language, it does allow us to communicate with routines that are written in 6502 machine or assembly language. Such routines are known as machine language subroutines."

Appendix F of the *BASIC Reference Manual* goes into the details on how to make a machine language subroutine and BASIC talk to each other.

### Approach

Getting back on track now, the problem we wish to solve is stated as follows:

> Develop a capability for making AIM 65 text files accessible to BASIC. One entire line of text should be passed to BASIC at a time. Lastly, BASIC should be informed when an end of file has been detected.

Note that from our earlier definitions, a line may be wholly contained in or may span a block. A key requirement that the subroutine must meet is the reconstruction of text lines when necessary. To satisfy all these requirements both the monitor subroutines and the BASIC USR function will be used.

Two AIM monitor subroutines which we chose for use in the machine language subroutine are:

WHEREI located at $E848

INALL located at $E993

These subroutines are described in the Users Guide. WHEREI asks the user what the current input device will be. Assuming that the user responds with 'T' (for audio tape in AIM format), WHEREI will then ask for the name of the file desired. It will then locate the file on the tape. INALL reads a character from the current input device. If the current input device is an audio tape, INALL will see to the tasks of properly handling lines. INALL will start and stop the tape recorder as necessary in order to obtain a complete line. Thus, two of our requirements are already solved.

Interfacing a machine language subroutine to BASIC is straightforward. The BASIC program simply has to poke the address of the machine language program into memory locations $04 and $05. The next step is to invoke the USR function. This will start up the machine language subroutine. The *BASIC Reference Manual* tells us how to pass a single numeric value to and from BASIC. We will use this feature to pass the line length and end of file indicator to BASIC.

There is one interface problem remaining. That is, how do we pass the text line from the machine language subroutine to BASIC? The USR function limits us to a numeric value. Well, we will be bold and make an assumption. Then we will design the subroutine to fit the assumption. Assume that the BASIC program has defined a character string variable named A$. Furthermore, assume that the A$ is 80 bytes long. We can then design the machine language subroutine so that it will locate A$ in BASIC's memory and store the text data there. If A$ is guaranteed to be 80 bytes long, we can be sure that text editor and BASIC lines can be read.

There are other approaches to reading these text files. For example, the USR function can be used to call WHEREI. The AIM 65 can then be put in the tape mode. At this point, the BASIC program can issue INPUT statements to read data directly from the tape. This approach is very simple and to the point. However, it suffers from two disadvantages. First of all, since the input device was changed to a tape, the keyboard is deactivated for the entire duration of the file read. This can be nasty, especailly if your program requires some input from the user as it is running. The second disadvantage is that the data on the tape must be in the proper format to be processed by the INPUT statement. This means that there must be commas between values and that string data may need to be enclosed in quotation marks.

At the expense of a machine language subroutine, we have developed a method of reading AIM text files which is completely general. Any text file, including BASIC source programs, can now be read with BASIC. We have addressed the problems mentioned above. The AIM 65 is put in the tape mode only as long as it takes to read one line. The data on the tape can be in any format - you do not have to worry about commas and quotation marks.

### Loading the Subroutine

Although our listings show that the subroutine is located at $7C00, the subroutine is completely position-independent. This means that you can put it anywhere in memory that you like. You will not have to change a single byte of code. Of course, you will have to remember where you put it because BASIC will need to know.

The hex dump in Figure 1 is probably easier to work with when initially entering the machine code. If you prefer to enter the code in instruction format using Figure 2, just be careful of the absolute addresses which appear as branch operands. For ease of future use, you will probably want to store the machine code on tape. Thereafter, the subroutine can be loaded with the 'L' monitor command.

When bringing up BASIC, be sure to respond properly to the MEMORY SIZE question. Respond with the difference of the number of bytes of RAM in your system minus 164 bytes for the subroutine. For example, MEMORY SIZE in a 4K system would be 4096 – 164 or 3932.

### Procedure

We hope that the subroutine has been put together so that it is easy

to use. Only three steps are required to read AIM 65 text files:

1. Open the file.
2. Invoke the USR function.
3. Test the USR function return code.

**Step 1 - Open the File**

A file is opened by zeroing memory location $F5 (245 decimal). This causes the subroutine to invoke WHEREI in the AIM monitor. In BASIC we open a file as follows:

```
10 POKE 245,0
```

If you intend to read more than one file in the same BASIC program, you must open each one of them at the appropriate time with a POKE statement. Only one file can be open at a time.

**Step 2 - Invoke the USR Function**

One text line or record will be returned to the BASIC program each time the USR function is used. We will illustrate this in BASIC:

```
20 A$ = ""
30 FOR I = 1 TO 80
40 A$ = A$ + "*"
50 NEXT
60 POKE 4,0
70 POKE 5,124
80 L = USR (0)
```

Lines 20 through 50 set up A$ as an 80 byte character string in accordance with our design criteria. If the BASIC program does not alter the length of A$ during subsequent processing, these lines could be moved to the section of the BASIC program that opens the file. The important thing to remember is that the subroutine will insist that A$ is 80 bytes long — no more or no less.

The contents of A$ prior to calling the subroutine, however, do not matter. Before giving you any data, the subroutine will always blank out A$. Thus, you are guaranteed not to encounter any data left over from a previous line.

Lines 60 and 70 are very important! They tell BASIC where the machine language subroutine is located. Line 60 POKEs the low order byte of the address (expressed in decimal) into memory location $04. Similarly, line 70 POKEs the high order byte of the address into memory location $05. In our example, the machine language subroutine is located at $7C00. Make sure you tailor lines 60 and 70 for your system.

If this is the only machine language program that your BASIC program is using, the two POKEs may also be included as part of the file opening logic.

Finally, line 80 invokes the USR function. This causes BASIC to call our machine language program. We are not passing a value to the machine language subroutine.The 0 is just a dummy argument. The machine language subroutine will read the next text line from tape and give it back to us is A$. BASIC will resume processing with the next statement after line 80.

**Step 3**
**Test the USR Function Return Code**

In line 80, the USR function passed a value back to the variable **L**. We call this value a return code. It can be assigned to any numeric variable - it doesn't have to be **L**. The value of the return code tells us the status of the read operation.

a. Return code is less than 0
If the return code is negative, this means that an error condition has been detected. Probable error conditions are that A$ was undefined or not 80 bytes long. (The AIM monitor worries about catching read errors.)

b. Return code is equal to 0
The return code will be set to zero when end of file is reached. No special action is required to "close" the file as it is done automatically.

c. Return code is greater than 0
A successful read operation will be signalled by a return code which is greater than zero. Furthermore, the return code will tell you the actual number of data bytes which were stored in A$. In other words, it will tell you the line length.

WARNING: Under no circumstances should another read be executed after end of file has been detected. If this should happen, you may have to hit the reset switch to regain control.

We might finish our example this way:

```
90 IF L < 0 THEN STOP
100 IF L = 0 THEN PRINT
"DONE":END
110 PRINT LEFT$(A$,L)
120 GOTO 80
```

Lines 90 and 100 terminate the program on an end of file or error condition respectively. Line 110 prints the text line. Line 120 branches back to read the next text line.

**Summing It Up**

Our sample program is printed in its entirety in Figure 3. Make a couple test files with the text editor. Run the test files through our sample program. You should see the lines of data that you entered printing out one by one. If you encounter any problems, go back and check the machine code carefully. Make sure that you've POKEd $04 and $05 with the correct address.

We hope that this capability to read text files adds a new dimension to your computing.

*Figure 1*

```
M>=7C00 AD 12 A4 48
< > 7C04 A5 75 85 F0
< > 7C08 A5 76 85 F1
< > 7C0C A5 77 C5 F0
< > 7C10 D0 12 A5 78
< > 7C14 C5 F1 D0 0C
< > 7C18 A0 FF A2 FF
< > 7C1C 68 8D 12 A4
< > 7C20 8A 6C 08 B0
< > 7C24 A0 00 B1 F0
< > 7C28 C9 41 D0 07
< > 7C2C C8 B1 F0 C9
< > 7C30 80 F0 0D 18
< > 7C34 A5 F0 69 07
< > 7C38 85 F0 90 D0
< > 7C3C E6 F1 D0 CC
< > 7C40 A0 02 B1 F0
< > 7C44 99 F0 00 C8
< > 7C48 C0 05 D0 F6
< > 7C4C A4 F2 C0 50
< > 7C50 D0 C6 88 A9
```

```
< > 7C54 20 91 F3 88
< > 7C58 10 FB A5 F5
< > 7C5C D0 08 20 48
< > 7C60 E8 AD 12 A4
< > 7C64 85 F6 A0 00
< > 7C68 A5 F6 8D 12
< > 7C6C A4 20 93 E9
< > 7C70 C9 0A F0 F9
< > 7C74 C9 0D D0 0A
< > 7C78 C5 F5 85 F5
< > 7C7C F0 0B A2 00
< > 7C80 F0 9A 91 F3
< > 7C84 85 F5 C8 D0
< > 7C88 DF A0 00 AD
< > 7C8C 34 A4 D0 0A
< > 7C90 AD 00 A8 09
< > 7C94 10 8D 00 A8
< > 7C98 D0 E4 AD 00
< > 7C9C A8 09 20 8D
< > 7CA0 00 A8 D0 DA
<
```

### Subroutine Logic

We've included in this section a technical description of how the machine language subroutine operates. This should give you enough information to modify the subroutine to fit your particular needs.

Figure 4 depicts the logic of the machine language subroutine. The logic is described through the use of Warnier-Orr diagrams. Readers who are not familiar with these diagrams should refer to the December '77, January '78, and March '79 issues of *BYTE*. Very basically, Warnier-Orr diagrams are interpreted as follows. The sequence in which operations are performed is given by reading from the top of the diagram to the bottom. The hierarchy of functions flows from left to right. As we go through the actual subroutine logic, the power of this design technique will become more apparent.

Figure 5 summarizes the use of zero page variables. These locations are shared with the text editor. However, since the text editor and BASIC do not operate concurrently, there is no conflict.

Upon entry to the subroutine, an AIM monitor variable INFLG is saved on the stack. INFLG tells AIM what the current input device is. Since the subroutine will change the input device to audio tape, we have to be careful here not to lose track of input devices. The next task is to examine BASIC's symbol table to determine if A$ has been defined as an 80 byte character string according to our design assumptions. In either case, the logic will proceed to a next lower hierarchical level. This is indicated by the next sets of braces to the right. When control is returned back to the first level, INFLG is restored from the stack. Most often, this will again put the AIM in the keyboard mode. Finally, the subroutine passes a return code to BASIC. The 16 bit integer return code in registers A,Y (MSB, LSB) is given to BASIC by a JMP indirect to location $B008 in the BASIC ROM.

*Figure 2*

```
K>*=7C00
/40
7C00 AD LDA A412      Save INFLG
7C03 48 PHA
7C04 A5 LDA 75        Start of BASIC's symbol table
7C06 85 STA F0
7C08 A5 LDA 76
7C0A 85 STA F1
7C0C A5 LDA 77        Reached end of symbol table?
7C0E C5 CMP F0
7C10 D0 BNE 7C24      No...
7C12 A5 LDA 78
7C14 C5 CMP F1
7C16 D0 BNE 7C24      No...
7C18 A0 LDY #FF       Error exit - set return code to -1
7C1A A2 LDX #FF
7C1C 68 PLA           Normal exit
7C1D 8D STA A412      Restore INFLG
7C20 8A TXA
7C21 6C JMP (B008)    Return to BASIC
7C24 A0 LDY #00
7C26 B1 LDA (F0),Y
7C28 C9 CMP #41       Have we found A$?
7C2A D0 BNE 7C33
7C2C C8 INY
7C2D B1 LDA (F0),Y
7C2F C9 CMP #80
7C31 F0 BEQ 7C40
7C33 18 CLC           Point to next symbol table entry
7C34 A5 LDA F0
7C36 69 ADC #07
7C38 85 STA F0
7C3A 90 BCC 7C0C
7C3C E6 INC F1
7C3E D0 BNE 7C0C
7C40 A0 LDY #02       Found A$...
7C42 B1 LDA (F0),Y    Get address and length of A$
7C44 99 STA 00F0,Y
7C47 C8 INY
7C48 C0 CPY #05
7C4A D0 BNE 7C42
7C4C A4 LDY F2
```

Assuming A$ satisfies the design assumptions, the subroutine will set A$ to blanks. This is done every time the subroutine is called. Next a counter which counts the number of data characters read is zeroed. Then a test is performed to determine if the subroutine is being called for the first time. (NOTE: the sucess of this test relies on the BASIC program to POKE location $F5 to 0.) INFLG is next restored from a temporary variable at $F6. The AIM should now be configured to accept input from audio tape. So then the character read routine is called repeatedly until a carriage return is detected and processed.

If A$ does not meet our design assumptions, the return code is set to −1. This should alert the BASIC program of an error condition.

IF the subroutine is being called for the first time, the AIM subroutine WHEREI is invoked. WHEREI issues the familiar prompt:

OUT =

Normally the user responds with "T". The AIM monitor will then prompt for the file name and tape drive number. When WHEREI finishes, INFLG, which was just set by WHEREI, will be stored in a temporary at $F6. This completes the initialization sequence.

*Figure 2 cont.*

```
K>*=7C4E
/40
7C4E C0 CPY #50     Is A$ 80 bytes long?
7C50 D0 BNE 7C18    No, then error
7C52 88 DEY         Yes, blank out A$
7C53 A9 LDA #20
7C55 91 STA (F3),Y
7C57 88 DEY
7C58 10 BPL 7C55
7C5A A5 LDA F5      Is it the first time called?
7C5C D0 BNE 7C66
7C5E 20 JSR E848    WHEREI
7C61 AD LDA A412    Store new INFLG in a temporary
7C64 85 STA F6
7C66 A0 LDY #00
7C68 A5 LDA F6      Restore INFLG from the temporary
7C6A 8D STA A412                          variable
7C6D 20 JSR E993    INALL
7C70 C9 CMP #0A     Ignore line feeds
7C72 F0 BEQ 7C6D
7C74 C9 CMP #0D     Is it a CR?
7C76 D0 BNE 7C82    No...
7C78 C5 CMP F5      Was previous char a CR?
7C7A 85 STA F5
7C7C F0 BEQ 7C89    Yes...
7C7E A2 LDX #00     End of text line
7C80 F0 BEQ 7C1C    Return to BASIC
7C82 91 STA (F3),Y Store the char in A$
7C84 85 STA F5
7C86 C8 INY
7C87 D0 BNE 7C68    Now go read the next char
7C89 A0 LDY #00     End of file...
7C8B AD LDA A434    Which tape drive are we using?
7C8E D0 BNE 7C9A
7C90 AD LDA A800    Turn drive 1 on
7C93 09 ORA #10
7C95 8D STA A800
7C98 D0 BNE 7C7E    Exit
7C9A AD LDA A800    Turn drive 2 on
7C9D 09 ORA #20
7C9F 8D STA A800
7CA2 D0 BNE 7C7E    Exit
```

*Figure 3*

```
LIST

 10 POKE 245,0
 20 A$ = ""
 30 FOR I = 1 TO 80
 40 A$ = A$ + "*"
 50 NEXT
 60 POKE 4,0
 70 POKE 5,124
 80 L = USR(0)
 90 IF L < 0 THEN STOP
 100 IF L = 0 THEN PRINT
          "DONE": END
 110 PRINT LEFT$(A$,L)
 120 GOTO 80
```

WARNING: Locations 4 and 5 must be POKEd with the physical address of the machine language subroutine. In this program the subroutine is at $7C00.

The read character routine calls a lower level read routine until a character other than a line feed is found. The purpose for skipping line feeds, is to facilitate the reading of BASIC source program tapes. (BASIC prefixes each source program line with a line feed.) One of two lower level routines is then invoked depending on whether the character just obtained is a carriage return or not.

The lowest level read character routine is simply an invocation of the subroutine INALL. INALL will obtain a character from the current input device.

If the character obtained is a carriage return, the previously read character is examined. If the current character is not a carriage return, the current character is stored in the next available byte of A$ (pointed to

by $F3 and $F4). The count of the number of characters read is updated.

If the current and previous characters are both carriage returns, end of file has been detected. The proper tape drive is turned back on (INALL turned it off) so the tape can be rewound. Then the return code is set to 0.

If the current character is a carriage return, but the previous character was not, the end of a line has been reached. The return code is set to the count of the number of characters read. Note: the carriage return is neither counted not stored in A$.

*μ*

*Christopher Flynn became interested in microcomputers when he assembled a MITS Altair computer kit in 1976. Since then, he has obtained a KIM-1 and an AIM-65. His KIM system interfaces to several S-100 boards by means of a KIMSI Motherboard.*

*The AIM is his favorite system. It has 32K of RAM and uses a Model 33 teletype for hardcopy output. His software interests include Assembly language and BASIC.*

*Applications developed on the KIM and AIM range from an interpreter to a home budgeting and accounting system. To support this hobby, Chris is employed by the Fairfax County government as a systems analyst for the county's tax systems.*

*Christopher's wife, Nancy, has learned to program in BASIC. Their two year old daughter, Becky, when asked what her father's name is, has been known to respond, "6-5-0-2".*

***Figure 4***

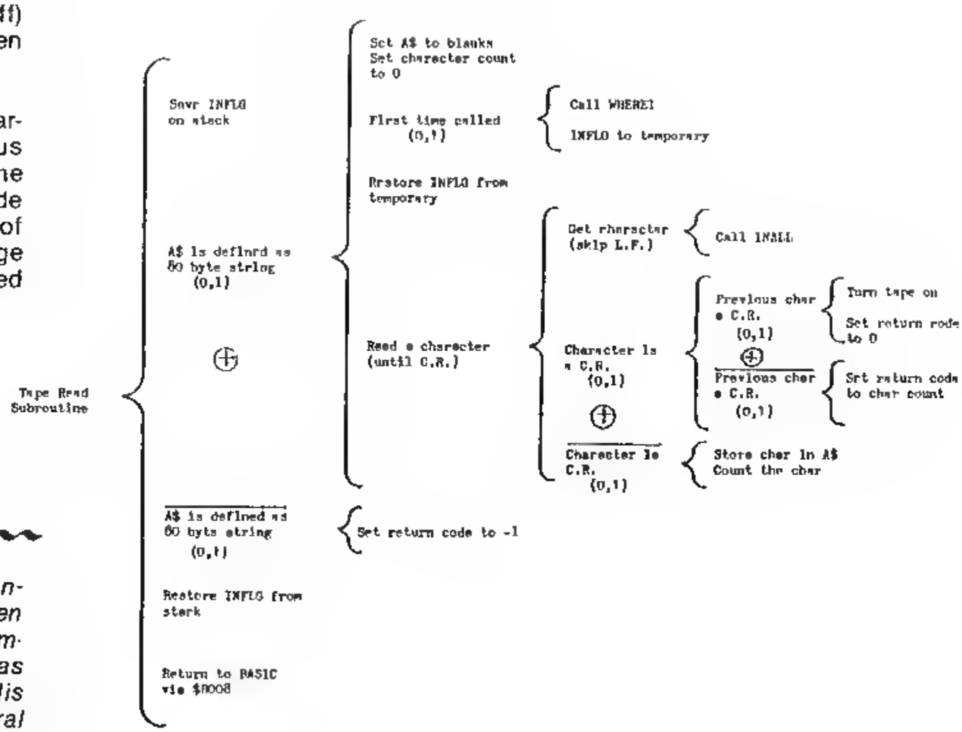



***Figure 5***

| | | |
|---|---|---|
| SYMTAB | $F0,$F1 | Pointer to BASIC symbol table |
| LEN | $F2 | Length of A$ |
| APNT | $F3,$F4 | Pointer to A$ in BASIC's memory |
| TEMP | $F5 | First time switch; hold area |
| TINFLG | $F6 | INFLG hold area |

# MICRO Club Circuit

Here is yet another installment of 6502-related clubs. We continue to be encouraged by the terrific response to our request for new clubs. Now we have so many that we can't print them all in a single two-page listing!

If you have registered with us and you are not presented here, do not be dismayed. Next month you will be first on the list! The mail has just been loaded with club information.

Those of you who are listed please take a moment to make sure that the information is correct. Notify us of any errors. Up-dates should be sent to us periodically.

Does your club publish a newsletter? Do you need advertiser's? Want to exchange an ad? If the answer to any of these questions is yes, then let us know!

To become an officially registered club please send for the correct form. This is the only way to get a free one year subscription for your club's library. Have your club listed to increase your membership.

Address any information or requests to:

**MICRO Club Circuit**
P.O. Box 6502
Chelmsford, MA 01824

**Western Educational Computing Conference, San Diego, California**

November 20, 21

The theme of the seminar/exhibit is "Educational Computing in the '80's" and will feature papers and seminars on the use of computing in higher education for instruction, administration, and research. Luncheon speakers will be Capt. Grace Hopper, USN, and Bernard Luscombe, President, Coastline College.
For further information contact:

Ron Langley
Director, Computer Center
California State University
1250 Bellflower Boulevard
Long Beach, California
90840

**Texas A&M Micro Computer Club**

This club meets every two weeks on Wednesday nights. Conrad G. Walton Jr. is the President of 80 members. He can be contacted at:

Box M-9
Aggieland Station, TX
77844

*"The club owns 2 8K Pets and one SWTPC 6800 system with Pencom disk. Aim to provide education for the community in the applications and use of micro-computers."*

**Forth Interest Group**

This educational club asserts that their world-wide membership is 950. They meet on the fourth Saturday of the month. They list no contact person but the address for their club is

P.O.Box 1105
San Carlos, CA 94070

**Apple Information and Data Exchange**

Meets on the second Tuesday of each month at:

Computer Corner
1800 S. Georgia
Amarillo, TX 79109

George Johnson is the President of AIDE. Theiraddress is:

5700 Dixon
Amarillo, TX 79109

*"Mutual aid and sharing of information."*

**Apple Puget Sound Program**

Meets on the second Tuesday of each month. Over 3000 members. Dick Hubert is the President. A.P.P.L.E. Library Exchange. Contact:

Fred Merchant, Sec.
517-11th Avenue East
Seattle, WA 98102

*"Assists its members in the use and understanding of the Apple Computer. One time Apple-Cation fee and annual dues."*

**Madison Pet User's Club**

Meets on the first Thursday of the month at 7:30 pm in the Washington Square Building. Membership around 50. Contact:

B.A. Stewart
501 Willow
W. Baraboo, WI 53913

*"Exchange Information."*

**New England Computer Society**

Meets on the first Wednesday of the month at the Mitre Corporation Cafeteria in Bedford, MA. Robert Waite is the President over 200 members. Contact:

David Mitton, Sec.
P.O.Box 198
Bedford, Mass. 01730

*"General purpose, personal/hobby computing, technical information sharing."*

**San Francisco Apple Core**

Meets first Saturday of the month. Randy Fields is President. Membership of over 800. Contact:

Randy Fields
P.O. Box 4816
San Francisco, CA 94101

**Winnipeg Apple Computer Group**

Meets on the first Thursday of each month at 7:30 in the Computerland Store. Acting President is Mike Flood. Membership is still growing — over 30 currently. Contact Mike at:

5-1730 Taylor
Winnipeg, Manitoba
Canada, R3N 0N8

*"Increase members knowledge of programming, hardware, and data processing. Newsletter."*

**Burlington Micro Club**

Meets on the last Wednesday of the month at 7:30 pm at various locations. William Morris, President over 25 members. Contat him at:

67 Moxley Drive
Hamilton, Ontario
Canada, L8T 3Y8

*"Membership is open to everyone. Micro user '80, a club newsletter."*

**6502 Comp-Club**

Meets at various places. Members and those interested are notified through the mail as to the monthly arrangements. Robert Wilson is

club President. Over 25 members. For current information contact:

R. Wilson
Box 6007
Lawrenceville, N.J. 08648

*"Purpose: To consumate interest and to further knowledge of 6502 computers."*

**Erie Apple Crunchers**

Rudy A. Guy Is President over this newly organized group of 25 avid users. Contact them for more information:

P.O. Box 1575
Erie, PA 16507

*"Membership Is open to all Apple or Bell & Howell Apple owners or users. Developing a software library and we are willing to exchange software with other individuals or groups."*

**N.I.C.H.E.**
**Northern Indiana Computer Hobbyist**

Meets in the South Bend area on the last Monday of almost every month. Contact:

Eric Bean
927 S 26 Street
South Bend, In 46615

*"The meetings are open to all computer hobbyists, but is dominated by PETs."*

**Apple Byter's Computer Club**

Information regarding this club should be requested from S.E. Grove, Pres., Mail Station 33, Bldg R-19

H.E.S.E.A.
Hughes Aircraft
2060 Imperial
El Sugundo, CA 90245

*"A private club for Hughes Employees only but open to guests. Education of members in the use of computers by programmers and others. Buy at group rates, exchange software in public domain, and member of the I.A.C. (international Apple Core) Grow with others in the Greatest Hobby EVER!"*

**UPDATES—UPDATES—UPDATES**

**OSIO**

Washington, VA, MD group meets the first Tuesday of each month. Meets at the Walter Johnson High School In Rockville, Md. Contact:

Wallace Kendall, Pres.
9002 Dunloggin Road
Ellicott City, MD 21043

*"Study, advance, and promote the application of computers; publish newsletters; sponsor conferences, workshops, symposia, demonstrations, and publications on computers, etc."*

**Apples British Columbia Computer Society**

Meets first Wednesday of every month at 7:30. Various locations. Gary Little is President for 95 members. Contact him at:

101-2044 West Third Ave
Vancouver, B. C. Canada
V6J 1L5

*"All members are Apple II owners, aim is to discuss software and hardware."*

**Apple Sac**

Meets on the first Tuesday and third Wednesday of each month, with Assembly language classes on the third Tuesday. Bill Norris is president. 80 plus members. Contact:

Jerry Jewell
Computerland of
Sacramento
1537 Howe Avenue
Sacramento, Ca 95825

*"Fun, education, social, sharing of ideas and programs."*

# The MICRO Software Catalog: XXII

## Software announcements for the 6502 based systems

Mike Rowe
P.O. Box 6502
Chelmsford, MA 01824

Name: **ALGEBRA**
System: **PET 2001**
Memory: **8K or mora**
Language: **BASIC, Machina**

Description: A series of 7 programs (on one cassette) designed to assist a student through various levels of the subject. Topics include: Set operations, signed arithmetic, linear equations, factoring, and quadratic equations. An example of each class of problem is given, followed by a changing sequence of problems to be solved by the student. After each problem, as answer is provided to check results. Other Pet software available.

Copies: **New Ralaasa**
Price: **$19.95**
Author: **Lan Bugal**
Available: TYCOM Associates
68 Velma Avenue
Pittsfield, MA 01201

Name: **Compufar Station Singla Disk Copy**
System: **Appla II or Appla II Plus**
Memory: **32K**
Language: **Integer Basic or Applasoft**
Hardware: **Appla II, Disk II**

Description: Program will copy a complete diskette using an Apple II with only a single disk drive. The program will function properly on an Apple II or Apple II Plus with or without the Applesoft ROM Card or the Language System. It will run with DOS 3.1, DOS 3.2, or DOS 3.2.1 and will run on either a 32K or 48K system. On a 32K system it will take five passes for a full diskette while only three on with 48K. Requires a maximum of 3 passes on a 48K system, does verification, will initialize if desired and is faster than Apple's two disk copy.

Price: **$29.95, $2.00 s&h IL rasidants add 5 % salas tax.**
Includes: Disketta, phamphlet
Author: **Joal Upchurch**
Available: Computer Station
12 Crossroads Plaza
Granite City, IL 62040

Name: **AMATEUR RADIO COMMUNICATIONS PACKAGE**
System: **Appla II, Plus**
Memory: **18K**
Language: **Intarger**
Hardware: **Radcom Plus Card (suppliad), Disk II**

Description: Send-Reveive RTTY and Morse Code. Interface installs in Slot 2. Active bandpass filters. FSK output. Narrow Shift (170 HZ). LED tuning indicators. Scope monitoring. Computer grade circuit board. Gold plated contacts. Assembled and tested. Baudot speeds continuous 32 to 300 Baud. ASCII to 1200 Baud. Morse Code speeds 2 to 125 WPM. Split screen, receive, Xmit and Xmit buffer. Save text from a buffer to the Disk. Load text from Disk to a buffer (TX/RX). Display current system status or catalog. Normal/Invert RTTY Rx key control. Stored massages to limit of RAM. Much more!

Copies: **Just ralaasad**
Price: **$190.00**
Includes: Radcom Plus = Card, Software on Disk, doc.
Authors: **Radcom Plus Card by Alax M. Massimo AF6W**
**Software by Dr. Chris H. Galfo WB4-JMD**
Available: Alex M. Massimo
4041 41st Street
San Diego, CA 92105

Name: **Tha Craativity Lifa Dynamic Packaga**
System: **Appla II**
Memory: **48K**
Language: **Applasoff, Machina**
Hardware: **Appla II, Disk II**

Description: Draw, Write Music, Write Poetry! Draw Circles, elipses, triangles, frames, enclosuras, fireworks, squares, etc. (many more!) all at the touch of a key or two (without hitting return). Fill or partially fill any of the above figures to create an infinite variety of figures. Change to and from Regressive & Symmetry Modes. Write Music using your keyboard like a piano. Watch your notes be named and written on a cleff. Easily change pitches and durations. Write a poem. Choose 1, 2, or 3 forms, save and play later! MUCH MORE!!

Copies: **Many**
Price: **$19.95**
Includes: Disk, 88 page Prog. Manual, 2 drawing cards.
Author: **Avant-Garda Creafions**
**P.O.Box 30161 MCC**
**Eugana, OR 97403**

Name: **GAF Soffwara Utility Packagas 1 & 2**
System: **Appla II, Plus**
Memory: 32K
Language: **Intagar, Applasoft**
Hardware: **Appla wifh Disk II**

Description: A collection of useful utility programs. Utility 1: File Compare, a program that allows comparing of two versions of a program and reporting all differences to your

screen, printer, or disk file. Menu, a general purpose HELLO program that allows one keystroke program execution. Reads any size catalog to produce menu. Applesoft & Integer Sorts, fast implementation of Shell-Metzner sort can be adapted to your programs. Convert-To-Text, turns Applsesoft end Integer programs into text files. Utility Package 2 Includes Multiple Disk Catalog, File Cabinet Fast Sort, File Copy and Food Plan.

Copies: **Just released**
Price: **$30.00 each**
**$50.00 both**
Author: **Gary A. Foote**
Available: GAF Software
127 Mt. Spring Road
Tolland, CT 06084

Name: **LCMOD for Pascal**
System: **Apple**
Hardware: **Apple Language System**

Description: Allows DIRECT entry of upper/lower case into the Pascal Editor using the Paymar LCA. Uses the ESC key for a shift key and the ESC key is now a Control Q to prevent accidental deletion of text. Also provides generation of left and right curly brackets for comment delimiters and an underline for VARs, program names and file names.

Price: **$30.00**
Available: Southeastern Software
7270 Culpepper Drive
New Orleans, LA 70126

Name: **MAG Files**
System: **Apple**
Hardware: **Disk II**

Description: Having trouble keeping track of all those magazine articles you read? Here is the answer. Enter them once and use the search modules to find them again either by title or subject code. Requires Applesoft II.

Price: **$18.00**
Available: Southeastern Software
7270 Culpepper Drive
New Orleans, LA 70126

Name: **Bad Buy Diskette**
System: **Apple**
Hardware: **Disk II**

Description: Of course it is a bad buy. If you had issues 2 through 11 of the Southeastern Software NEWSLETTER, you could type these programs in yourself. They are a mix of Integer, Applesoft II and assembly language programs and utilities.

Price: **$9.99**
Available: Southeaster Software
7270 Culpepper Dr.
New Orleans, LA 70126

Name: **Double Precision Floating Point for Applesoft**
System: **Apple II, Plus**
Memory: **32K**
Language: **AssemblyLanguage. Use with Applesoft Programs.**
Hardware: **Disk II**

Description: Provides 21 digit precision for Applesoft programs. Arithmetic expressions, as well as INPUT and PRINT are supported. Applesoft subroutines for the standard math functions are included. Nearly standard syntax is used, with the ampersand feature. Efficient and compact, only 2048 bytes. Loads itself beneath your Applesoft prog. Works with Applesoft ROM card, with Applesoft in the Language System, or with RAM Applesoft.

Copies: 25
Price: **$50.00**
Includes: Diskette, Reference Manual
Author: **Bob Sander-Cederlof**
Available: S-C Software
P.O.Box 5537
Richardson, TX 75080

Name: **Letter Perfect**
System: **Apple II, Plus**
Memory: **Min. 32K**
Language: **Machine**
Hardware: **Apple II, Plus/ 32K min/ Dan Paymar Lower Case.**

Description: A character orientated word processor. It supports proportional spacing and is capable of working with any printer type. It is user orientated and menu driven. Complete documentation. Supports: global and local searches, complete formating, full ASC II character set with lower case on video display, headers, footers, page numbering, complete formating within body of text, top margin, and much more! Full cursor control.

Author: **Kenneth Leonhardt**
Available: LJK Enterprises, Inc.
P.O.Box 10827
St. Louis,MO 63129

Name: **Gus's Disk Utility**
System: **Apple II**
Memory: **16K, 32K, 48K**
Language: **Machine**
Hardware: **Apple II, Disk II**

Description: Program is designed to be an easy to use aid to working with the Apple II DOS 3.1 or DOS 3.2. Restore those accidentially deleted files, remove DOS from your diskette for more room on your data only disks, read/write to any sector, print file attributes (catalogs your disk and allows to choose any file on the diskette to give you file type, track sector list, the sector lists which contains your program), prints binary program parameters, and will map the free sectors of your diskette. Allows individual byte or sectors to be changed or transfered to another diskette.

Copies: **Just released**
Price: **$45.00**
Author: **Ralph D. Gustafson**
Available: Rainy City Software
4360 SW Parkview
Portland, OR 97225

Name: **Disk Apple II Report Textwriter - DART**
System: **Apple II Or Apple II Plus**
Memory: **32K**
Language: **Applesoft II**
Hardware: **Disk II, optional printer and lower case adapter**

Description: A program which composes reports, articles, letters and other documents, utilizing text files generated by the "DOS Text Editor". Text may be input in free form format, without regard to line length or pagination. Retrieves the data from

the file, formats it into lines of desired length, and displays it on a printer or Apple CRT. Changing the text requires only that the text file be modified with EDIT·II, and DART called to format and output a new report. The variable input funcion allows form letters and standard text to be modified from the keyboard to produce custom letters and reports. File chaining allows an unlimited amount of input text.

Price: **$19.95 plus $1.25 s & h. Package speciel: EDIT·II end DART $37.89**
Copies: **Just released**
Includes: Diskette, user manual, and documentation
Author: **Robert Stein**
Available: Services Unique,Inc.
2441 Rolling View Dr.
Dayton, Ohio 45431

Name: **Disk Text Editor· Edit II**
System: **Apple II or Apple II Plus**
Memory: **Minimum 24K**
Language: **Applesoft BASIC**
Hardware: **Apple, disk end optional printer end lower cese edapter.**

Description: An improved version of the DDS Text Editor, designed to create and facilitate changes to disk files, reports, lists, etc. Also supports the cassette as a file device. Includes 35 commands. String commands allow searching, changing, and listing of single records or blocks of records for a specified word or phrase. User input. File commands merge input from various files, parts of files and text buffers. Handles full upper and lower case ouput to print devices. Works with DART.

Copies: **Over 200 of Edit·I**
Price: **Cessette $19.95**
**Diskette $23.95**
**Shipping $1.95**
Includes: User manual and documentation
Author: **Robert A Stein, Jr.**
Available: Apple Computer Stores or
Services Unique, Inc.
2441 Rolling View Dr.
Dayton, DH 45431

Name: **Progrem Writer**
System: **Apple**
Memory: **32K minimum**
Language: **Applesoft**
Hardware: **1 Disk Drive**

Description: This program was written to speed up the process of writing advanced business program. It works as a data management system, but also writes disk stetements as permanent line number, if requested. Supports 20 fields per entry, searching or sorting by any field, generating reports, packing numbers to increase disk space, plus many more. Use for inventory, checks, phone bumbers, etc. Simple to use with instructions.

Price: **$29.95**
Copies: **Just releesed**
Includes: Diskette, instructions,examples
Author: **Wilford Niepreschk**
Available: Wilford Niepraschk
5921 Thurston Avenue
Virginia Beach, Va 23455

Name: **Visible Memory Routlnes**
System: **8K PET**
Memory: **2K**
Language: **Machine Lenguage**
Hardware: **8K PET, MTU Visible Memory Boerd**

Description: Machine language software easily accessable by BASIC. Package includes clear screen, plot·a·point, line draw, and RDSE plotting programs. Dther programs available to run with VM Routines: VM LISS·3D space Art, VM Sprirals, Hi·resolution spirals, VM 3D Plots, same 3D images as seen in many ads. More coming. Send SASE for list of these and other programs. Copies of MTU user's Notes available.

Copies: **Just released**
Price: **$7.95 for VM Routines**
Includes: Cassette, Documentation
Author: **Russell A. Grokett, Jr.**
Available: Pet Library
401 Monument Road
Jacksonville, FL 32211

Name: **PSA/1**
System: **Apple II, Plus**
Memory: **16K**
Language: **Applesoft Besic**
Hardware: **Apple II (Printer, opt)**

Description: A cassette·based introduction to computer scheduling. Using critical·path scheduling techniques, it allows the user to define a project, input time estimates for each job in the project, and then compute schedules for each job. Computes the earliest and latest each job can be started, finished, in order to meet deadlines. Also schedules delays without harm to other jobs. Displayed on video.

Copies: **New Releese**
Price: **$25.00 (WA edd 5 %)**
Includes: Cassette, User Manual
Author: **Don Teylor**
Available: Express Marketing
21866 Clear Creek Road
P.O.Box 1736/MSC
Poulsbo, WA 98370

Name: **Files**
System: **Apple II 3.2 or 3.2.1 DOS**
Memory: **32K min.**
Language: **Applesoft**
Hardware: **Disk necessery, Printer optionel**

Description: File is a modular File utility program which is designed to allow the user to build files, add to existing files, correct records, delete, lock, unlock, insert records, move records, delete records, find records, sort, eppend files together, rename and save files, and view file data.

Copies: **Just releesed**
Price: **$49.95**
Includes: Disk and manual
Author: **Merc Goldferb**
55 Pardee Place
New Haven, Conn. 06515

*While we have been lenient in the past regarding the length of the entries in the Software Catalog, we must now insist that future entries be kept as brief as possible. We think that twelve to fifteen lines in the "description" part of the entry should keep it about right. The other parts, as long as needed.*

*We now have so many entries backed up, that we feel this policy is only fair to give everyone 'equal time'. We will be fored to edit, or return any entries that we judge too long.*

*Mike Rowe*

# 6502 Bibliography: Part XXII

**Continuing bibliography of 6502 related material**

Dr.William R. Dial
438 Roslyn Avenue
Akron, OH 44320

**642. Applesaad (Softsida) (Jan. 1980)**

Mickius, Lance and Summers, Murray, "Dog Star Adventure," pg. 36-48.
Rescue the Princess Leya.

643. **Craative Computing 6, No. 1 (Jan. 1980)**

Howerton, Christopher, "Grandapple Clock," pg. 104-107.
Now your Apple can tick, chime, and keep time.

Carpenter, Chuck, "Apple-Cart," pg. 134-137.
Discusses Keyword search, the MOD function, New Apple products, etc.

Yob, Gregory, "Personal Electronic Transactions," pg. 148-150
Discusses short utility routines, a programming for formatting numbers, etc.

**644. SYM—PHYSIS, Iss. 1 (Jan/Fab. 1980)**

Anon., "2KSA Assembler/Editor," pg. 3-6.
Assembly language program for the Sym-1.

Anon., "Relocate for the SYM-1," pg. 7-12.
Machine language program for the Sym-1.

Gettys, Thomas, "MERGE/DELETE Program for SYM Basic," pg. 13-16.
Utility routine.

**645. Appla 1, No. 3 (1979)**

Willson, Dr. M. Joseph, "The Challenge to Personal Computers In Science and Industry," pg. 2-5.
The Apple II will be on board the Space Shuttle where it will monitor scientific experiments.

Anon., "Applications of the Apple," pg. 7-18.
Discussion of a number of applications including evaluating paramedic and hospital procedures, endocrine levels in the birth process, Pascal in Education, testing telephone lines, use in the trucking industry, prospecting by computer and use in military games in "think tanks."

**646. Racreational Computing 4, No. 1 (Jan. 1980)**

Mulder, David, "Merging on the PET," pg. 40-41.
Put two programs together with this routine.

**647. Racreational Computing 8, No. 1 (Jan/Fab 1980)**

Hall, David J., "Computing for Health and Equality," pg. 8-11.
DAll about Holistic Health and the PET.

Dellman, Tracy, "Holistic Computing - A Program Idea for Healthy Living," pg. 12-14.
A PET oriented program on holistic health.

Thornburg, David D., "The Presto-Digitizer Tablet," pg. 16-18.
A low cost alternative to data entry keyboards.

Sevik, Jim and Eric, "A Learning Program for Problem Readers," pg. 25-28.
A PET Program for readers with reading problems.

**648. Kilobaud Microcomputing No. 37 (Jan. 1980)**

Anon., "Ohio Scientific's Small Systems Journal," pg. 10-13.
Discusses the OSI-DMS Ouotation/Estimation System, the Educational System, Inventory Control, Purchasing System, and Bills of Material System.

Baker, Robert W., "PET Pourri," pg. 14-16.
Discusses New Pet Products, Axiom Printers, Programming Ideas and Tips.

Schmeltz, Leslie R., " 'Core' and More for Your Apple," pg. 110-114.
Accessories for your Apple.

Freeman, Robert, "The Metamorphosis of a 'Custom' PET," pg. 116-118.
Customize your PET.

Knapp, Jeff, "Darkroom Master," pg. 126-130.
Use your PET in the Darkroom.

**649. Stams From Appla 3, Iss. 1 (Jan. 1980)**

Stein, Dick, "PASCAL Time," pg. 7-14.
Three example programs which either reads or writes a data file.

**650. Kilobaud Microcomputing Iss. 38 (Feb. 1980)**

McCormack, Chris, "Microchess Modifications," pg. 68-69.
Enchance this game for your KIM.

Ramsey, David, "Two Intriquing and Usetul Apple II Peripherals," pg. 70-74.
Getting to know Speechlab and Apple Clock.

Sparks, Paul W., "Development of a Text-Handling Program: A Learning Experience," pg. 112-118.
Handling words on the PET.

Martellaro, John, "Apple's Hidden Floating-Point Routines," pg. 132-135.

Lightning-fast number crunching.

Spisich, John, "Add a Digital Tape Index Counter to the PET," pg. 158-160.
Construct this counter for your PET cassette and locate files quickly and accurately.

Blalock, John M., "A Printer for the KIM or SYM," pg. 186-192.
The Selectric finds another home.

**651. Creative Computing 6, No. 2 (Feb. 1980)**

Zimmerman, Mark, "Blackbox for the PET," pg. 112-117.
A game with graphics.

Carpenter, Chuck, "Apple-Cart," pg. 148-151.
Hints on using diskettes, Apple I/O Circuits, tips on using Pascal, Applesoft formatter.

**652. The Target (Jan/Feb. 1980)**

Bresson, Steve, "CHAIN," pg. 6-7.
Controlled loading and execution of multiple files from tape on the AIM 65.

**653. Call—Apple 3, No. 1 (Jan. 1980)**

Spurlock, Loy, "Creating a Hi-Res Character Set," pg. 13-15.
A Basic program for creating characters.

Hyde, Randall, "Assembler Maxi-Reviews," pg. 18-23.
Reviews of the Microproducts Assembler, the SC-Assembler II, ASM/65, EAT (Edit and Assemble Text), LIZA, UCSD Adaptable Assembler (Pascal).

Konzen, Neil, "ZOOM," pg. 28-32.
Two versions: one for Basic and one in assembly language.

**654. MICRO No. 21 (Feb. 1980)**

Peck, Robert A., "Expanding the SYM-1...Adding an ASCII Keyboard," pg. 5-7.
Fairly simple procedure.

Fam, Richard, "A HIRES Graph-Plotting Subroutine in Integer Basic for the Apple II," pg. 9-10.
A Basic subroutine is presented which permits graph plotting.

Morris, E.D., Jr., "Multiplexing PET's User Port," pg. 13-14.
Multiplex when you need to Input or Output more bits of data than your micro can handle.

Phillips, Robert, "The Binary Sort," pg. 15-16.
A concise description of the Binary Sort concept and an Implemenatation in Basic.

DeJong, Marvin L., "A Complete Morse Code Send/Receive Package for the Aim 65," pg. 19-26.
A valuable program for the Hams among the AIM users.

Swindell, Jack Robert, "The Great Superboard Speed-Up and Other RAMblings," pg. 31-32.
Here is all you need to make your OSI Model 600 board run twice as fast as it normally does.

Urban, Michael, "KIM-1 Tape Recorder Controller," pg. 35-39.
Some techniques for using a 6502 micro for controlling switches are presented, as for example, controlling a tape deck.

Tripp, Robert M., "Ask the Doctor," pg. 41-43.
Converting the SYM Tiny PILOT to work on KIM; Slow Display for the AIM; Chart of the AIM, SYM and KIM expansion pinouts.

Taylor, William L., "Graphics and the Challenger C1P, Part 3," pg. 47-53.
Third article shows how to put the pieces together.

Rowe, Mike (Staff), "The MICRO Software Cataloque XVII," pg. 55-56.
Nine New Programs for the 6502 micros.

Dial, William R., "6502 Bibliography: Part XVII," pg. 59-62.
Another 150 references are listed.

**655. BYTE, 5 No. 2 (Feb. 1980)**

Newcomb, Robert K., "Another Plotter to Toy With, Revisited," pg. 202-207
A plotter for the KIM.

**856. Personal Computing 4, No. 2 (Feb. 1980)**

Wheeler, Dwight, "Mechanical Paintbrush," pg. 56-57.
A graphics program for the PET.

**657. Interface Age 5, No. 3 (March 1980)**

Baker, Al, "Game Corner," pg. 36-42.
Time Trials is a new program for the Apple II.

Adler, Alfred, "The Micro-Mathematician," pg. 44-55.
A continuation of a Fourier Analysis program started earlier.

**658. SoftSide (AppleSeed) (Feb. 1980)**

Dubnoff, Jerry, "Supernim," pg. 10-15.
Adding a second dimension to this old Apple game.

Anon., "Elementary Math," pg. 22-23.
A lo-res graphics program with sound to assist in additon drills, Apple.

Brandon, Jack, "State Capitals," pg. 27-29.
An educational Apple game.

Anderson, Chip, "Connection," pg. 32-35.
A lo-res graphics program for the Apple.

Wagner, Roger, "Musical Scales," pg. 39-41.
A program to teach musical scales with the Apple.

Anon., "Sort," pg. 47.
A utility program for the Apple.

Anon., "The Vocal Apple," pg. 50-51.
Short Utility to make the Apple more vocal and responsive.

Anon., "Programming Tips for the Apple," pg. 54.
How to avoid unwanted blanks when editing PRINT statements.

**659. On Computing 1, No. 4 (Spring 1980)**

Williams, Gregg, "The Ohio Scientific C4PMF," pg. 39-45.
A review of a 6502 based microcomputer.

Hafner, Everett, "An Apple in Hanoi," pg. 70-78.
An interesting account of bringing up and maintaining a modern microcomputer in Southeast Asia.

**660. Dr. Dobb's Journal 5, Iss. 2 No. 42**

Brown, Dewitt S., "A User Interface to Apple II Program Renumbering," pg. 26-31.
Simplification of procedure for using renumbering routines.

Lindenschmitt, Gary, "Another Phone Dialer," pg. 43-44.
A phone dialer for the PET.

**661. Fort Worth Area Apple User Group Newsletter (Feb. 1980)**

Meador, Lee, "More About Interupts," pg. 1-4.
A tutorial on Interupts for the Apple.

Meador, Lee, "DOS Disassembly," pg. 4-9.
Third Installment of the Assembly listing of the Apple II DOS.

**662. 6502 Usar Notas, No 17**

Silvestri, Gino F., "Match This," pg. 1-3.
An Interactive game for the "naked" KIM.

Shljanowski, Rush, "How to Transfer Basic Programs from PET to KIM," pg. 4-5.
A utility program for the KIM, moditied by Eric Rehnke.

Deas, Glen, "Basic Cassette I/O Mods," pg. 5-7.
Mods for lead and save cassette routines for the KIM.

Doutre, Ben, "Tiny Basic," pg. 7-10.
Misc. notes on Tiny Basic toı the KIM.

Silvestri, Gino, F., "Broaden Your I/O Cheaply with a Non-6500 PIA," pg. 14.
How to use 8080 peripherals with the KIM.

Goıdon, H.T., "KIM Audible Warning Inteıface," pg. 15.
Software and hardware for an audible KIM.

Clements, W.C., JR, "Interfacing the TVT-2 Video Board with the KIM-1."
A How-to articled on interfacing.

Hogg, Frank, "Cassette Load Display on KIM LEDs," pg. 16-17.
Load Memory from Tape with display.

Hooper, Philip K., "Cassette Save Using Alternate Starting Address," pg. 17.
How to read a cassette file Into a memory block other than than the one trom which It was dumped.

Nelis, Jody, "AIM Info — AIM printer Moditication," pg. 20.
How to clear up wavy lines and how to get heavier print.

Bresson, Steve, "AIM 65 Basic — Data Save/Load Scheme," pg. 20.
Save and Load strings and data in text torm from Basic.

List, Gunnar, "TINY BASIC for SYM," pg. 22.
Tiny Basic modified for the SYM.

Regal, Ron, "OSI Notes," pg. 22.
Interfacing a 42K Ram board to the C-24P

Carlson, Edward, "Zero-Page Map for Basic in the C2-4P," pg. 23.
Memory Map.

Leasia, John D., "Pseudo Random Number Generatoı," pg. 24.
A utility routine.

Eaton, John, "KIMATH Support," pg. 24.
A routine to tind the Tangent of an angle.

Goenner, Markus, "Interrupt Routines and Breakpoint," pg. 24.
KIM-1 IRQ routine.

Jordan, Doug, "Square-Waver II," pg. 26.
A short routine for audio.

**664. Abacus II, Iss. 1 (Jan. 1980)**

Davis, James P., "Two Diamonds - A Puzzle Game," pg. 4-7.
A game adapted for Applesoft.

Avelar, Ed, "Remote Control for the Apple II," pg. 4-7.
Special hardware to turn on a remote Apple by phone.

Anon., "Nicer Menu," pg. 7.
Select trom your apple disk catalog by designating a letter.

Anon., "Auxilliary Key Board Plug," pg. 8-9.
A second Keyboard for the Apple is possible with this plug.

Anon., "Math Section I (Addition)," pg. 10.
An Addition program in Integer Basic for the Apple.

Wilkerson, David R., "Apple Writer 1.0," pg. 12-13.
A review of this word processor.

**665. Dr. Dobb's Journal 5, Issue 3 (March, 1980)**

Cason, R., "PET Tape CURE," pg. 43.

**666. Stems from Apple 3, Issue 2 (Fab. 1980)**

Hoggatt, Ken, "Ken's Korner—Basic and Pascal," pg. 3-7.
Similarities between Basic and Pascal.

Pteiffer, Jim, "How Applesoft Stores String Arrays," pg. 4-6.
Discussion ot String Arrays and how to clear such space.

Byerly, Kent, "Literal Input Fix," pg. 7.
A fix tor a program "Literal Input".

**667. Byte 5 No. 3 (March 1980)**

Helmers, Carl, "Hunting the Computerized Eclipse," pg. 6-12.
Use of the Apple in an Eclipse Monitoı operation.

Matthews, Randall S., "Hydrocarbon Molecule Constructor," pg. 156-166.
An Oıganic Chemistry teaching aid.

Couchman, James C., "KIM-1 Multiplication and Division," pg. 212-216.
Routines to multiply and divide two 16-bit signed quantities.

Hoopeı, Phillip K., "The Correct Order of Operations Can Shorten Code," pg. 242-244.
Pointer decrementing on the 6502.

**668. Nibble (Jan/Feb. 1980)**

Anon., "Hi-Resolution Multi Color Kaleidoscope," pg. 7.
Two Hi-res color programs for the Apple.

Anon., "Sort 'em Out," pg. 8.
Discussion of Sort techniques on the Apple.

Anon., "Initialize New Files Automatically with ONERR GOTO," pg. 9.
How to use the ONERR GOTO instruction on the Apple.

## Missing MICRO Information?

MICRO is devoted exclusively to the 6502. In addition, it is aimed at useful, reference type material, not just "fun and games". Each month MICRO publishes application notes, hardware and software tutorials, a continuing bibliography, software catalog, and so forth. Since MICRO contains lots of reference material and many useful program, most readers want to get the entire collection of MICRO. Since MICRO grew very rapidly, it quickly became impractical to reprint back issues for new subscribers. In order to make the older material available, collections of the reprints have been published.

[A limited number of back issues are still available from number 7 to 18 and 20 to current. There are no 19's left.]

**The BEST of MICRO Volume 1** contains all of the significant material from the first six issues of MICRO, from October/November 1977 through August/September 1978. This book form is 176 pages long, plus five removeable reference cards. The material is organized by microcomputer and almost every article is included. Only the ads and a few 'dated' articles have been omitted. [Now in third printing!]

**Surface...$7.00** **Air Mail...$10.00**

**The BEST of MICRO Volume 2** covers the second six issues, from October/November 1978 through May 1979. Organized by microcomputer, this volume is 224 pages long.

**Surface...$9.00** **Air Mail...$13.00**

**The BEST of MICRO Volume 3**, covering the twelve issues from June 1979 through May 1980, will be over 400 pages long. It is scheduled for late summer 1980. The price is still to be determined.

For a free copy of the Index for Volumes 1, 2, and 3, please send a self-addressed, stamped envelope to:

**BEST of MICRO, P.O. Box 6502, Chelmsford, MA 01824**

## Advertisers' Index

# Complete Your MICRO Collection

## NOW is the best time to complete your MICRO collection — during our Summer Special!

*Order back issues from now until September 30, 1980 and receive our special price. Complete your MICRO collection at these low prices.*

**Special prices on issue *numbers* 7 [Oct/Nov 1978] through 18 [Nov 1979].**
**Issues 7-12 Regularly $1.75 Surface Issues 13-18 Regularly $2.25 Surface**

**Summer Special Price: $1.00 for *any* issue 7 — 18 in the U.S.**
**$1.25 for *any* issue 7 — 18 foreign.**
**All back issues will be shipped Surface. Air Mail service only at regular rates.**

***Issues 20-24 available for $2.25, or check with your local dealer. Sorry, number 19 [Dec 1979] is no longer available!***

# Free Copy of MICRO

## Help spread the good word about MICRO — send a free copy to a friend during our Summer Special.

***During this period, you can share MICRO with a friend. Send in the address label from your MICRO, along with the name and address of a friend whom you think would enjoy receiving MICRO, and we will send him a free back issue. Back issue sent will be publisher's choice, and this free offer expires September 30, 1980.***

### Regular Subscription Information

One year's subscription = 12 Issues. US: $15.00, All other countries: $18.00 for surface. Air Mail rates for Central America: $27.00, Europe/S. America: $33.00, All other countries: $39.00.

Best of MICRO Voulme 1 (Issues 1-6): $7.00 for Surface, $10.00 for Air Mail. Best of MICRO Volume 2 (Issues 7-12): $9.00 Surface, $13.00 Air Mail. All of MICRO Volume 2: $9.00 Surface, $13.00 Air Mail.

All payments must be in US dollars. Please make checks payable to MICRO. Foreign payments in International Money Order or cash.

MICRO
P. O. Box 6502
Chelmsford, MA 01824
(617) 256-5515